



User's Guide

EDT Camera Link Simulator (CLS) Family



for PCI or PCI Express

Doc. 008-02781-02
2011 January 04

Engineering Design Team (EDT), Inc.

1400 NW Compton Drive, Suite 315

Beaverton, OR 97006

p 503-690-1234 / 800-435-4320

f 503-690-1243

www.edt.com

EDT™ and Engineering Design Team™ are trademarks of Engineering Design Team, Inc. All other trademarks, service marks, and copyrights are the property of their respective owners†.

© 1997-2011 Engineering Design Team, Inc. All rights reserved.

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., 20230, 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, USA) 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

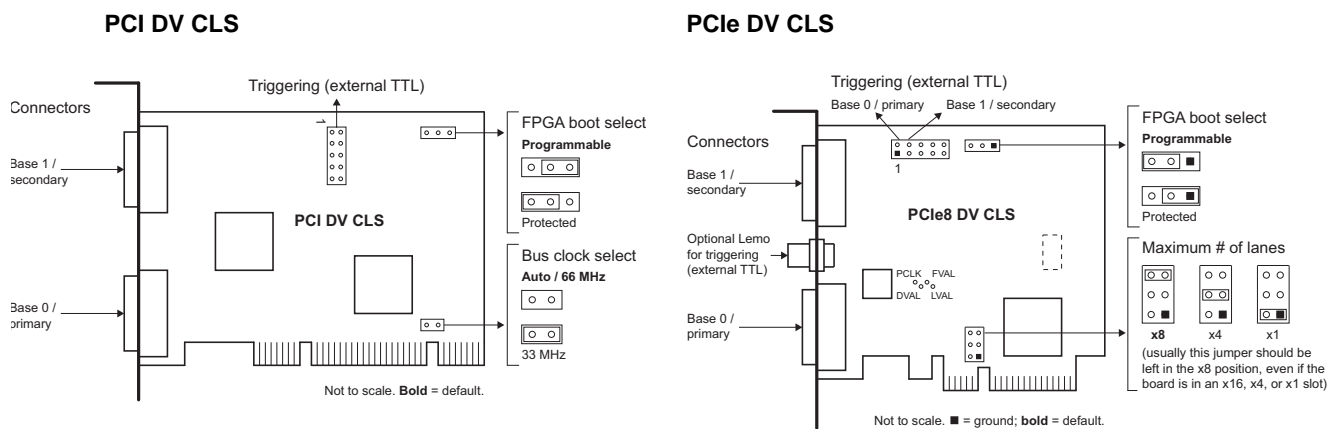
Overview.....	1
Related Resources	2
Requirements.....	2
PCI	2
PCI Express	3
Installation	4
Included Files.....	4
Building Applications.....	5
Cables.....	5
Operation.....	5
FPGA Configuration.....	5
Image Data Source	6
Serial Data	6
Triggering.....	7
About the Camera Link Standard	7
Initializing the Board	8
Initialization	9
Options for clsiminit.....	9
Simulator-specific Directives.....	10
Using the UART For Serial Communications.....	11
Sending Simulated Data.....	11
Image Data From Internal Counter	11
Image Data From Host Via DMA	12
send_tiffs image list format.....	13
send_tiffs code overview	13
Registers	15
Camera Link Registers	15
Simulator Registers.....	19
Timing Registers	21
Pinouts.....	24
Camera Link (MDR26) Connectors.....	24
Debug Connector.....	25
Appendix A: Interleaving Registers	29
Revision Log.....	31

EDT Camera Link Simulator (CLS) Family

Overview

EDT's Camera Link simulator (CLS) boards are designed to provide simulated Camera Link data. As shown in [Figure 1](#), the PCI version is for a PCI bus, while the PCIe version is for PCI Express.

Figure 1. EDT CLS Boards: PCI and PCIe versions



A CLS board works by sending simulated image data to a framegrabber board via Camera Link cabling attached to the MDR26 connectors on the boards. Base-mode simulation uses only the primary connector (the one nearest the bus) on each board, while medium- and full-mode simulation use both connectors.

CLS boards work with a wide variety of EDT and third-party products. High-level information about the CLS products and certain compatible EDT products is provided below.

NOTE For links to detailed datasheets (specifications) for all EDT products, see [Related Resources on page 2](#).

CLS product	Bus type	Camera modes simulated	Data rate
PCI DV CLS	PCI (32 bits / 66 MHz)	Base, medium	Up to 200 MB/s
PCIe8 DV CLS	PCI Express (8-lane)	Base, medium, full	Up to 800 MB/s

Other compatible EDT products

EDT Camera Link framegrabbers

For PCI, PCIe, PMC, cPCI, and fiberoptic applications

EDT Camera Link extenders

For long-range extension (up to 10 km) over fiberoptic cable

EDT record-and-playback systems

For data capture with up to ten EDT or third-party boards

Related Resources

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

EDT Resources

<i>Description</i>	<i>Detail</i>	<i>Web link</i>
• Product specifications (datasheets)	PCI DV CLS or PCIe DV CLS	www.edt.com (search by product)
• Other EDT digital video (DV) products	Framegrabbers, extenders, recorders	www.edt.com/digital_video.html
• DV Hardware Addendum	For hardware details	www.edt.com/manuals/pdv
• Camera Configuration Guide	For supported cameras	www.edt.com/manuals/pdv
• Application programming interface (API)	HTML & PDF	www.edt.com/manuals.html
• Installation packages: Windows, Linux, Solaris, Mac OS	Software / firmware	www.edt.com/software.html

Standards / Specifications

<i>Description</i>		<i>Web link</i>
• Camera Link	Camera Link standard	www.machinevisiononline.org

Parts

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

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

For the PCI DV CLS, the PCI interface runs at 66 MHz and can sustain a maximum DMA bandwidth of approximately 200 MBytes per second. You can use the PCI DV CLS in a bus slot of 66 MHz or faster – or in a 33 MHz slot, although in that case the maximum sustainable DMA bandwidth drops 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 framegrabber board 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 framegrabber so that they use different PCI buses.

NOTE If your image source is an internal counter, not actual images from the host (see [Sending Simulated Data on page 11](#)), the CLS is not using the PCI / PCIe bus and thus is not consuming bus bandwidth.

PCI Express

For the PCI Express version of the CLS (PCIe8 DV CLS), as for the PCI version, performance rates depend upon your system and its setup. Systems using EDT's PCI Express digital video products have tested at a maximum of 800 MBytes to 1.1 Gbytes per second.

(EDT's PCIe4 DV C-link uses 32-bit transfers, limiting its total maximum throughput to about 220 MB/sec.)

EDT's PCI Express 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 PCI Express 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.

Due to the location of the edge connector dividers, 8-lane PCI Express boards will not fit into 4-lane PCI Express slots. To get around this obstacle, some system manufacturers provide slots that are open at the back end, and EDT 8-lane boards should work in these slots. Another solution is to provide an 8- or 16-lane slot but wire it as 4-lane slot, thus working around the physical differences and allowing boards with different numbers of lanes to negotiate down to the lowest common denominator electrically. In any case, users should verify any non-standard configuration.

Installation

EDT provides installation packages for all supported operating systems (Windows, Linux, Solaris, Mac OS). 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 2](#).

To install the CLS:

1. Install the Pdv driver software as specified above.
2. Install the board into the computer according to your hardware manufacturer's instructions.
3. Cable the board to your framegrabber or other input device. For details on which cables are required for which boards, see [Cables on page 5](#).

Included Files

The CLS driver package includes a simulator setup application with a graphical user interface (GUI), as well as C source and executable for several command-line example, diagnostic, and utility programs.

Applications specific to the CLS include:

<code>clsim.bit</code>	FPGA configuration file for the PCI version of the CLS.
<code>clsiminit</code>	A command-line application to set up the CLS for a specific output. For the list of arguments and options, enter: <code>clsiminit --help</code>
<code>clsim</code>	A GUI for <code>clsiminit</code> . For help, see the online Help menu.
<code>genericssim.cfg</code>	Generic software initialization script that you can edit to configure the CLS for the camera you are simulating. Edit it to provide values for image height, width, and depth at least. For details, see Initializing the Board on page 8 .
<code>send_tiffs</code>	A command-line application to send one or more TIFF images to the framegrabber as simulated camera image data. For a discussion of using this application, see Sending Simulated Data on page 11 .
<code>clsim_lib</code>	C library file containing a set of functions specific to the CLS.
<code>clsim_lib.c</code>	C source for <code>clsim_lib</code> .
<code>clsim_lib.h</code>	Header file for <code>clsim_lib</code> .

C source is also available for the applications `clsiminit` and `send_tiffs`.

Building Applications

CLS files are in the distribution directory `/opt/EDTpdv` (for Unix systems) or `C:\EDT\pdv` (for Windows).

The package includes both executables and C source code for all of the examples, diagnostics, and utilities. To rebuild any of these programs, simply follow the steps below, using an appropriate compiler and the `make` (Unix) or `nmake` (Windows) application.

1. Run Pdv Utilities (Windows) or navigate to the installation directory in a terminal window (Unix).
2. Do any of the following:

- To build an application, enter...

```
make file
```

...replacing *file* with the name of the example program you wish to build.

- To build and install all the example programs, enter:

```
make
```

- Alternatively, to build the sample programs, set up a project in Windows Visual C++.

Cables

Connect your input device to your EDT CLS board with a standard Camera Link cable. For part information, see the link under [Related Resources on page 2](#). For connector pinouts, see [Pinouts on page 24](#).

Base-mode devices must cable the input device to the bottom connector — the one closest to the PCI or PCIe bus connector. Medium-mode input devices use both connectors.

Operation

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.

FPGA Configuration

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 applications `clsiminit` and `clsim`.

Alternatively, the command-line application `bitload` can be used to load `clsiminit.bit` (e.g., `bitload clsiminit.bit`).

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.

After loading the firmware, the UI FPGA registers must be configured with the specifications of the camera you wish to simulate – for example, pixel clock rate, image width and height, and number of taps. Camera configuration scripts (`.cfg`, such as `genericsim.cfg`) offer a convenient way of doing this, as well as passing other information on to the EDT driver and library routines. You can then run your application program, which can call EDT driver routines to send DMA data to the CLS board.

NOTE For details on CLS FPGA configuration, see the link to the EDT DV Family User's Guide under [Related Resources on page 2](#).

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 framegrabber 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.

Though primarily targeting cameras with one, two, three, or four 8-bit taps, the PCI DV CLS can be configured to simulate most base- and medium-mode Camera Link cameras of up to 32 bits per pixel clock cycle. For example, you can simulate a 12-bit camera by running in two-tap mode, taking 16 bits from the DMA stream for each pixel clock cycle.

The PCI Express 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).

Serial Data

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 modified for each frame.

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_x` library functions (where *x* represents any of several different functions). The UART transmitter and receiver are implemented as a shifter and holding register, without associated FIFOs. Host interrupt response time must be sufficient to read any data byte received before the next byte is completed.

Triggering

The CLS can be configured to trigger each frame-valid signal from the framegrabber 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.

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 2](#)).

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 framegrabber, 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 an additional 24 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 PCI Express version of the CLS), the UART and camera control wires in the second cable can provide an additional 24 image data lines beyond medium mode, for a total of 72 data bits – although the Camera Link specification describes cameras no larger than 64 bits.

Initializing the Board

As much as possible considering its different purpose, the CLS has been designed to work like an EDT Camera Link framegrabber, and the software described here has been designed to work in a manner similar to `initcam`. EDT camera configuration directives and digital video library functions that can be used for a camera simulator are available to you for your own applications.

The `clsiminit` program is the main example program, as well as the application that initializes and configures the CLS. (Alternatively, you can use the graphical application `clsim` to initialize and configure the CLS; it offers all the same functionality.)

- On the PCI Express version of the CLS, the UI FPGA is embedded.
- On the PCI version of the CLS, `clsiminit` loads the FPGA configuration file `clsim.bit` into the UI FPGA and sets up the simulator based on the specified camera configuration file.

For example...

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

...sets up the simulator according to the directives specified in the `genericsim.cfg` configuration file.

Camera configuration files are editable text files. To use `genericsim.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 camera configuration file already exists, you can use that camera configuration file instead; the `clsiminit` application reads the same camera configuration files as `initcam`, the application that initializes the EDT framegrabber for various cameras. The camera configuration files provide values for such parameters as image dimensions and number of taps. For details on `initcam` and camera configuration files, refer to the EDT DV Family User's Guide and the Camera Configuration Guide (see links under [Related Resources on page 2](#)).

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 Directives on page 10](#)). You can add these simulator-specific directives to any camera configuration file — either `genericsim.cfg`, or your chosen camera configuration. If you later use the edited camera configuration file to set up an EDT framegrabber for your chosen camera, `initcam` ignores the simulator-specific directives; camera configuration proceeds just as it would if you had not added them. You can therefore use the same camera configuration file in both instances.

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

Just as `initcam` does, `clsiminit` works by filling in the `PdvDependent` structure, which is then 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 and bits on the board.

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

Initialization

The basic method for running `clsiminit` is to specify a camera configuration file:

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

For example, to simulate the Redlake Megaplug II ES 11000 camera, enter:

```
clsiminit -f camera_config/es11000cs.cfg
```

Doing so will configure the CLS to simulate a camera with an image size of 4008 x 2672 (12-bit) pixels.

If you have more than one board using the PDV driver installed in your host, supply a unit number to specify which one you wish to initialize:

```
clsiminit -u unit_number -f camera_config/file
```

By default, the board in the first PCI or PCIe bus slot is unit 0.

NOTE For a list of all boards (with unit numbers) installed in the host, you can run EDT's `pciload` application. For details, refer to the EDT DV Family user's guide (see link under [Related Resources on page 2](#)).

An invocation such as either of the above works with any eight-bit Camera Link configuration file, using defaults for clock speed and frame blanking.

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.

<code>-u unit</code>	The unit number of the CLS (by default, 0).
<code>-B</code>	Do not load the FPGA configuration file.
<code>-q</code>	Disables program output (quiet mode).
<code>-s</code>	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.
<code>-C</code>	Sets the first word of the frame to be <i>framecount</i> .
<code>-F freq</code>	Sets the pixel clock in MHz; takes a floating point argument (by default, 20.0).
<code>-t #taps</code>	Sets the number of taps to <i>#taps</i> .
<code>-f config_file</code>	Use the specified camera configuration file to set image parameters.
<code>-w width</code>	Image width (by default, 1024). Overridden by values specified in a camera configuration file specified with <i>-f</i> .
<code>-h height</code>	Image height (by default, 1024). Overridden by values specified in a camera configuration file specified with <i>-f</i> .
<code>-d depth</code>	Image depth (by default, 8). Overridden by values specified in a camera configuration file specified with <i>-f</i> .
<code>-v vblank</code>	Sets the interval between frames in lines (by default, 400).
<code>-V VcntMax</code>	Sets the total number of lines in the frame (ignored if <i>vblank</i> is specified instead with <i>-v</i>).
<code>-g hblank</code>	Sets the number of pixel clock cycles of horizontal blanking (by default, 300).

<code>-H <i>HcntMax</i></code>	Sets the total number of pixel clock cycles per line, including blanking (ignored if <i>hblank</i> is specified instead with <code>-g</code>).
<code>-r</code>	Reset the board. Zeroes all settings.
<code>--help</code>	Prints complete and current list of all directives and arguments.

Simulator-specific 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 framegrabber for a specific camera, as simulator-specific directives will be ignored.)

```
cls_pixel_clock:frequency in megahertz
```

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

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

<code>cls_hgap</code>	extra clock cycles per line to add to defined image width
<code>cls_hcntmax</code>	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.

<code>cls_vgap</code>	lines between active video
<code>cls_vcmtmax</code>	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`:

<code>cls_hfvstart</code>	where frame valid starts on first line
<code>cls_hfvend</code>	where frame valid ends on last line
<code>cls_hlvstart</code>	where line valid starts relative to frame
<code>cls_hlvend</code>	where line valid ends
<code>cls_hrvstart</code>	where DMA data starts
<code>cls_hrvend</code>	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 interleaving, see [Appendices on page 28](#), which shows how to simulate multi-tap cameras and how to use negative values to simulate cameras that implement taps moving from right to left.

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

Using the UART For Serial Communications

The application `clsiminit` initializes serial communications on the CLS. With the configuration file directive `cls_uartloop` set to 1, enabling loopback, the simulator echoes any serial characters that come in the UART receiver, sending them unchanged out the UART transmitter.

Test serial communication using `pdvterm`, a terminal emulator supplied along with the driver, and an EDT DV-series framegrabber.

For example:

```
pdvterm
> IDN? (from framegrabber)
IDN? (echoed response from simulator)
>
```

Sending Simulated Data

On the CLS, simulated data can come from an internal counter, or from the host via DMA. In all cases, 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 Camera Link simulator transmits only the 24 last significant bits of these 32-bit data words to the framegrabber. Thus, the first word of each frame received by a base-mode

framegrabber 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

Alternatively, you can send image data from the host via DMA. 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. Although `send_tiffs` supports only TIF images, EDT provides source code so you can modify the application to support other file formats if needed.

Before running `send_tiffs`, you must first initialize the CLS using `clsiminit` or `clsim`. For details, see [Initializing the Board on page 8](#).

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` (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 13](#)), but it can be a simple list of file names, 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 The EDT application `pciload`, included with the driver, lists all boards (with unit numbers) installed in a given host. See [Related Resources on page 2](#) for a link to the EDT DV Family User's Guide.

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

```
send_tiffs --help

-u unit           The unit number of the CLS (by default 0).
-N #buffers       The number of ring buffers to use. We recommend using four.
-i file           The input file with list of images and directives — by default, a file named
                  imagelist in the current directory, but this option allows you to specify another file.
-t #images       number of images to send (must be no larger than the number found in image list),
                  used if you have a large number of images in the image list but wish to send only
                  some of them.
-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` 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 the `#` — `send_tiffs`, then ignores the rest of the line. Otherwise, the first string of characters up to a space character specifies the name of a TIFF 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 hexadecimal. To use hexadecimal, 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`. 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:

```
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
```

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 application programming interface (see [Related Resources on page 2](#)).

The first thing `send_tiffs` does is to parse the command line arguments, get the list of images, and perform 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 we 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.

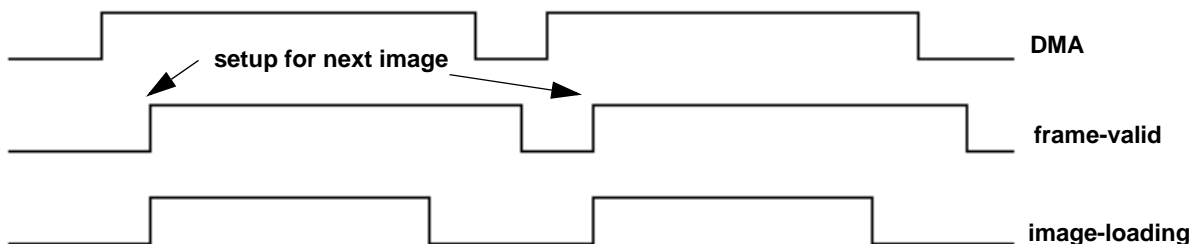
The setup required in [step 2](#) uses the EDT library Event system to ensure that an image is never missed:

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

This code registers the function `setup_clsimg_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_clsimg_event` does.

[Figure 2](#) 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 2. Timing of DMA, Image Setup, and Image-loading



DMA starts with the program's call to `edt_start_buffers(pdv_p, 1)`. After the simulator has 16 KB of data, it starts sending data out to the framegrabber. At the same time it sends the FVAL interrupt to the driver, which in turn calls `setup_clsimg_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.

Registers

The FPGA configuration file `clsim.bit` in the PCI version of the CLS (or the embedded equivalent in the PCI Express version) defines 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.

NOTE Address 0x05 is reserved and always returns 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 framegrabber.
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.

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 framegrabber.

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 framegrabber using the SERTFG UART signal. Data from the framegrabber 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: <ul style="list-style-type: none"> 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 <i>short 2</i> is transferred before <i>short 1</i> . Does not change the order of the bits within each short. See Figure 3 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 3 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 3. 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: $(20\text{MHz} / (\text{baud_rate} * 16)) - 2$ 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	<p>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).</p> <p>When clear, frame-valid determines which rasters have active data.</p>
6	LVCONT	<p>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.</p>
5	RVEN	<p>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.</p> <p>When clear, the entire raster is filled with valid data.</p>
4	UARTLOOP	<p>When set, serial data emitted by the framegrabber is echoed back unchanged, enabling a test of the framegrabber's serial port.</p> <p>When clear, your camera simulator application could respond to serial commands over the UART.</p>
3	SMALLOK	<p>When set, simulator starts DMA when 1 KB of data is in FIFO, allowing the simulator to handle images smaller than 16 KB.</p> <p>When clear, simulator waits until 16 KB of data is in FIFO before starting DMA.</p>
2	INTLVEN	<p>When set, enables four-tap interleaving — the four-tap reordering of 8-bit pixel values — using the Appendices - that is, 0x60–61 Tap 0 Start through 0x6E–6F Tap 3 Delta. Image data destined for the framegrabber 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).</p> <p>When clear, interleaving is off.</p>
1	FIRSTFC	<p>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.</p> <p>When clear, the first word is DMA or counter data.</p> <p>The FIFO_RESET bit in 0x02 Configuration initializes the framecount to 0x0001.</p>
0	DATAcnt	<p>When set, image data comes from the counters instead of the DMA stream.</p> <p>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.</p> <p>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.</p> <p>When set, also setting SMALLOK stops the simulator at the start of the next frame, to enable getting a single frame of counter data.</p> <p>When clear, simulator uses DMA data.</p>

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	<p>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.</p> <p>When DVSKIP is zero, data valid is always high, except during blanking.</p> <p>DVSKIP does not effect the timing of line-valid, frame-valid, or image data sent to the framegrabber. Therefore, a DVSKIP value of 7 causes only one-eighth of the DMA data to be captured by the framegrabber as valid image data; the rest is ignored.</p>
3–0	DV MODE	<p>Camera manufacturers exhibit no consensus on how to treat the data-valid signal. DVMODE allows most implementations to be simulated:</p> <p>00 = data-valid low during blanking.</p> <p>01 = data-valid high during blanking.</p> <p>10 = data-valid is treated the same during blanking as during active video (so DVSKIP also affects blanking).</p> <p>11 = data-valid always low (may change in future revisions).</p>

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 [0x4A–4B Vertical Active](#), after which it clears to zero. Therefore, a value of 0x04FF in [0x4A–4B Vertical Active](#) 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 framegrabber 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 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]	<p>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.</p> <p>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).</p> <p>If RVEN=0 in 0x40 Camera Link Configuration A, 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.</p>

Pinouts

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

Camera Link (MDR26) Connectors

The Camera Link (MDR26) connector pinouts for various modes are shown below.

Camera (or simulator) end	Frame-grabber end	Base mode Camera Link signal (primary connector)	Medium mode Camera Link signal (secondary connector)	Full mode Camera Link signal (secondary connector)
1	1	inner shield	inner shield	inner shield
14	14	inner shield	inner shield	inner shield
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	inner shield	inner shield
26	26	inner shield	inner shield	inner shield

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 number assignments and test points are shown below.

PINS:										TEST POINTS:			
2	4	6	8	10	12	14	16	18	20	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		
1	3	5	7	9	11	13	15	17	19	4	frame valid		
										5	data valid		
										6	pixel clock		
										7–20	[undefined]		

The PCI Express version of the board has the same **DEBUG** connector, but it is fully configurable. In addition, near the center of the PCI Express 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 1](#) on [page 1](#).

Appendices

Appendix A: Interleaving Registers

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

- Image data destined for the framegrabber 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 1. Interleaving: Two- and four-tap simulation

Two taps (to fill from starting points)			Four taps			Two taps (to fill from both ends)		
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:			Similarly, to emulate a four-tap model, use these starting points and deltas:			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		Start	Delta		Start	Delta
Tap 0	0	1	Tap 0	0	2	Tap 0	0	+2
Tap 1	512	1	Tap 1	512	2	Tap 1	1023	-2
			Tap 2	1	2	Tap 2	1	+2
			Tap 3	513	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.

0x60–61 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.

Revision Log

Below is a history of modifications to this guide.

Date	Rev	By	Pp	Detail
20110104	01	PH,RH	All	Added PCIe version of CLS board.
20070000	00	LW	All	Created new guide.