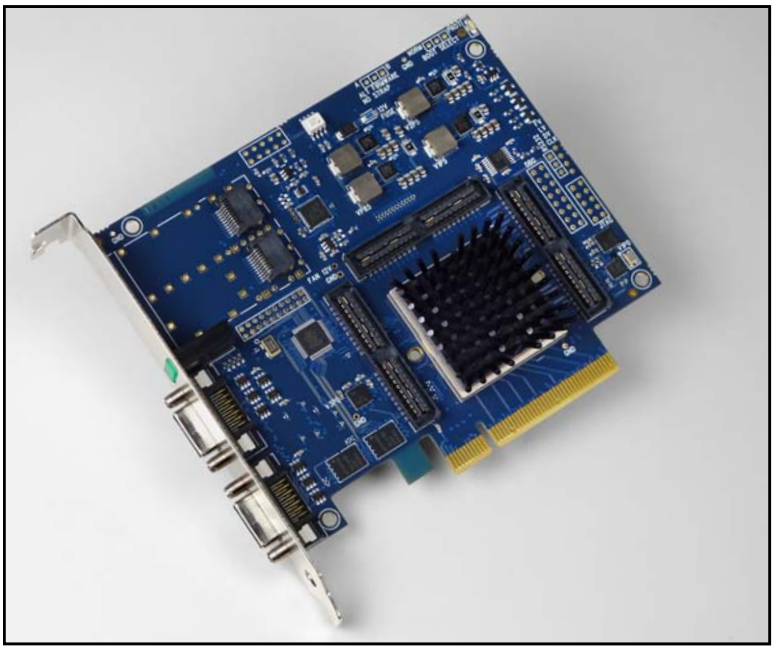
User's Guide

VisionLink CLS

Camera Link Simulator for PCI Express



Date: 2018 February 26
Rev#: 0002

EDT | Engineering Design Team, Inc.
3423 NE John Olsen Ave
Hillsboro, OR 97124
U.S.A.
Tel: +1-503-690-1234 | Toll free (in U.S.A.): 800-435-4320
Fax: +1-503-690-1243
www.edt.com

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 on such laws and regulations, see the website for...

- U.S. Department of Commerce, Bureau of Industry and Security
  https://www.commerce.gov/bureau-industry-and-security

...or, if ITAR status is indicated in the product’s documentation (on the title page or near the beginning), see the website for...

- U.S. Department of State, Bureau of Political-Military (PM) Affairs, Directorate of Defense Trade Controls (DDTC)
  https://www.pmddtc.state.gov/regulations_laws/itar.html

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

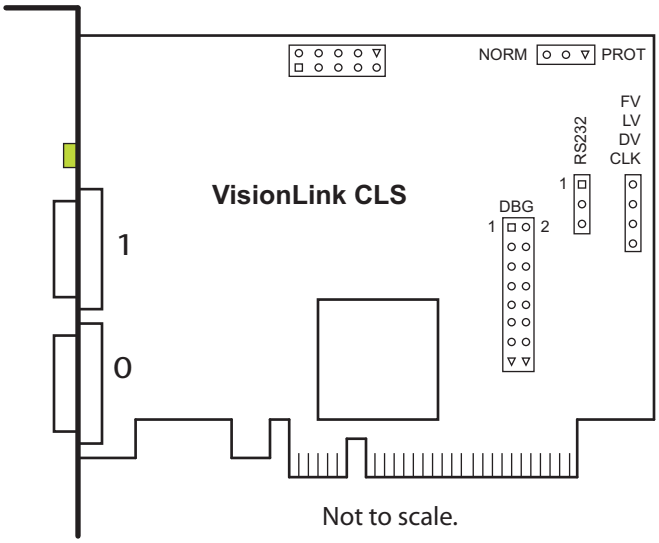
VisionLink CLS Camera Link simulator

Overview

The VisionLink CLS is a Camera Link simulator that simulates base through 80-bit mode cameras, 1–10 taps, at pixel clock rates of 20 to 85 MHz on a Gen3 PCI Express board. Actual or generated images are output as Camera Link data, facilitating development and testing of imaging systems, cameras, and software.

Images are sent via DMA from host memory, with data rates of up to 850 MB/s (85 MHz, 10-tap, 8-bit) as supported by the host computer. Internal counters can be used as an alternative source of image data.

Figure 1. VisionLink CLS



Requirements

To ensure reliable operation, your system must support sufficient PCIe bandwidth for your EDT product and for the camera you will simulate.

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/sec.

To reduce the likelihood of data dropouts due to bandwidth saturation, select a Gen3 PCIe system and avoid loading it down with other high-bandwidth processes or devices.

The VisionLink CLS will work in a 1-, 8-, or 16-lane slot, but will work at its rated bandwidth only in an 8- or 16-lane slot.

NOTE If your image source is an internal counter, not the host computer (see Sending Image Data From Host Via DMA on page 11), the CLS is not using the 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	*URL*
• VisionLink CLS product page	edt.com/product/visionlink-cls/
• Frame grabbers	edt.com/product-lines/camera-link-frame-grabbers/
• Fiber-optic extenders	edt.com/product-lines/camera-link-extenders/
• Addendum: Device Configuration Guide	edt.com/downloads/DeviceConfig/
• Application programming interface (API)	edt.com/api/index.html
• Installation packages: Windows, Linux	edt.com/file-category/pdv/
• Cabling	edt.com/clink-cables/

Standards / Specifications

Description	*URL*
• Camera Link standard	visiononline.org

Installation

EDT provides installation packages for Windows and Linux. These packages are provided on the EDT installation CD that ships with your EDT product, and are available online, at edt.com.

NOTE The VisionLink CLS requires version 5.5.3.9 or newer of the EDT driver package and libraries. Typically, applications developed with earlier EDT simulator boards (for example, the PCIe4 DVa CLS) will work for the VisionLink CLS with little or no modification to the source code; however they will need to be re-compiled and re-linked with the version 5.5.3.9 or newer libraries and headers. Applications thus updated will remain backwards compatible with earlier EDT simulator boards.

To install the VisionLink CLS:

1. Remove any previous version of the EDT Vision (PDV) driver software
2. Install the appropriate PDV driver software package as specified above.
3. Install the board into the host computer according to the computer manufacturer's instructions.
4. 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 2. For connector pinouts, see Appendix B: Pin Assignments on page 17.

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`, `send_tiffs` and `pdvterm`, while the EDT user's guide for frame grabbers (see Related Resources on page 2) 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 a VisionLink CLS (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).

NOTE Run `pciload` with no arguments to see how the system has enumerated your devices, then adjust the `-u` and `-pdvN` arguments in the following commands accordingly.

To begin, connect channel 0 of the frame grabber to channel 0 of the simulator.

From a terminal window, run the following commands:

```
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 #(loop through sending images in txt file)
```

In another terminal window:

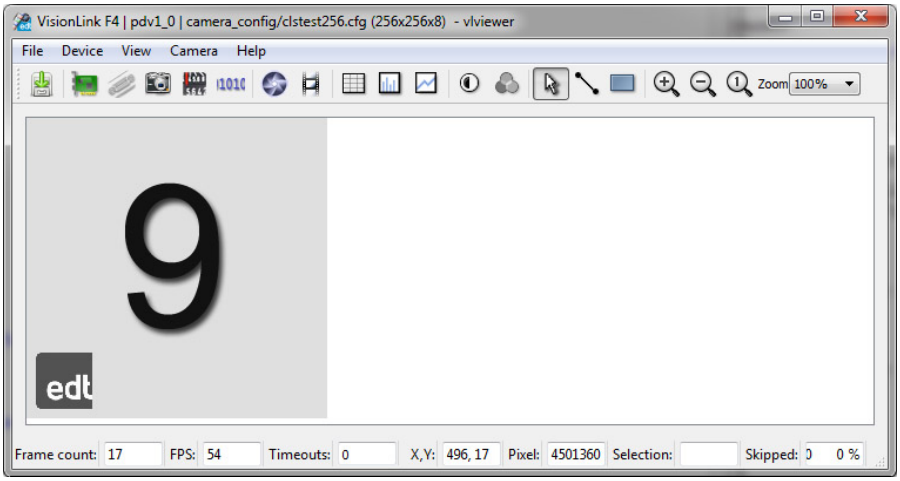
```
take -u 0 -N 4 -l 100 #capture 100 images from the EDT framegrabber
```

If the output shows 0 timeouts, everything is working. You can add a `-b <file>` argument to cause `take` to save the images in sequentially numbered files, for viewing via your favorite image viewer.

To display images captured in real time from an EDTframegrabber, you can invoke the vlviewer GUI via the desktop icon or by running...

```
vlviewer -pdv0
```

Selecting Single or Continuous capture will sequentially display the image files sent out from the simulator; in this example each image being sent has one digit, 0-9.



Simple Serial Verification

To test serial communication, use the loopback option (`-l`) to `clsiminit`, which enables serial 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")
```

To test serial using an EDT framegrabber, initialize the simulator and framegrabber using `clsiminit` and `initcam`, respectively, then run `pdvterm -u <unit>` in separate terminal windows, one for the CLS and one for the framegrabber. Any text typed in to the CLS instance of pdvterm should be reflected in the framegrabber instance, and vice-versa.

Operational Details

The VisionLink CLS provides the following key features:

- 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

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 `simple_clsend` or `send_tiffs` application, or your custom application.

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 over the PCI express bus 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 12kB] 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 / 0xCC–CE Vertical Count Maximum (the value in those registers, minus the active video time specified in the 0x4A–4B / 0xCA–CB 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.

In some applications, the number of lines in each image can vary. Before the frame is started, the line count is written into 0x4A–4B / 0xCA–CB 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 / 0xC4 FillA, 0x46 / 0xC6 FillB, 0x4C–4E / 0xCC–CE Vertical Count Maximum, 0x58–59 / 0xD8–D9 Horizontal Read Valid Start, and 0x5A–5B / 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 VisionLink 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, command-line application `pdvterm`, which is included in installation package.

Triggering

The VisionLink 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 VisionLink CLS to trigger each line-valid signal on either the rising or falling edge of the CC1 line.

For more details on triggering, consult the EDT user's guide for your frame grabber (see Related Resources on page 2).

Units, Connectors, and Channels

Units, connectors, and channels are defined as follows...

unit	EDT board (simulator or framegrabber)
connector	physical connector (SDR or MDR)
channel	DMA channel

The VisionLink CLS has two SDR26 connectors.

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

In medium, full, or 80-bit mode, each device requires two connectors on the EDT board. In this case, the two connectors work together to support one DMA channel.

On boot, the system assigns a unique number to each EDT board found, starting with 0 (the sequence is system-dependent). The software provides a unique handle to represent each unit / connector combination.

Running `pciload` with no arguments will display a list of all boards (by unit number) installed in the host. For a complete description of pciload, consult the user's guide for your frame grabber (see Related Resources on page 2).

EDT example and utility applications use the `-u` *`unit`* `and -c` *`channel`* arguments to specify the unit and channel, respectively (defaults are unit 0, channel 0). For example, to initialize unit 1 (the second board) and connector 1 (the secondary connector), enter...

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

In the EDT API, the unit and channel are selected via parameters in the `pdv_open...()` subruoutines, which then return a pointer to the structure (handle) that represents the opened device (unit and DMA channel). For example,

```
PdvDev *pdv_p = pdv_open_channel("pdv", 0, 0);
```

Each unit / channel combination can be opened and manipulated independently using the handle returned by the open subroutine. For details, consult the EDT API (see Related Resources on page 2).

The simulator's backpanel has four LEDs numbered 0, 1, 2, 3 (left to right), to display channel status, as shown below

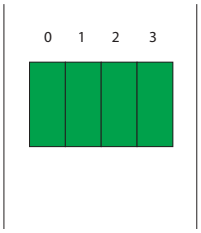


Table 1. LED behavior and significance

LED #	Status
0	Connector (channel) 0 pixel clock: on = present; off = absent
1	Future use
2	Connector (channel) 1 pixel clock: on = present; off = absent
3	Future use

Software

The following software resources are provided for use with the VisionLink CLS

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 specified products. Files specific to EDT simulator boards include a simulator setup application, as well as C source and executables for several command-line example, diagnostic, and utility programs. These simulator-specific files are listed in the table below.

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

Table 2. CLS-specific files

File	Description
`clstest256.cfg`	Sample configuration file for demonstration, testing, or verification via `simple_clsend` or `send_tiffs` (located under `camera_config/`).
`visionlink_cls.bit`	FPGA configuration file for the VisionLink CLS.
`clsim_lib.c`, `clsim_lib.h`	C source and header file for `clsim_lib` – the part of the PCI DV library (`pdvlib`) that covers EDT simulator boards.
`clsiminit`, `clsiminit.c`	A command-line application (binary and source code) to set up the board for a specific output. For the list of arguments and options, enter... `clsiminit --help`
`clsimvar`, `clsimvar.c`	Utility to test line-scan operation (binary and source code); also lets you vary the output line length. For usage, enter... `clsimvar --help`
`generic*cl.cfg`	Generic files (in `camera_config` directory) that you can copy and edit to configure the board for the camera you are simulating. Provide at least values for image height, width, and depth. For details, see Software on page 7.
`imagelist.txt`	Sample image list for demonstration, testing, or verification (via `simple_clsend` or `send_tiffs`).
`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 Image Data From Host Via DMA on page 11.
`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 less-used functions stripped out. Currently it works for TIF image files only, but has stubs for other image formats to help users add their own image formats. Though not all that simple, `simple_clsend` is simpler as example code than `send_tiffs`. For details on using this application, see Sending Image Data From Host Via DMA on page 11.
`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 2]. Files installed on Windows have the extensions `.lib` and `.dll`; files installed on Linux have the extensions `.so` and `.a`.

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; exceptions – such as differences between Windows and Linux – 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 apply to 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 2).

Building or Rebuilding an Application

By default, EDT's installation package will install files in `C:\EDT\pdv` (Windows) or `/opt/EDTpdv` (Linux). 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 2008 or later)

Use the included Visual Studio 2008 project, or set up your own project in Visual C++. For newer versions of Visual Studio, use the Visual Studio migration tool to update the provided solution / project files.

To build from the command line, from `C:\EDT\pdv` (the installation directory), run *Pdv Utilities* and use the `nmake` utility. First make sure your build environment variables are set properly. With Visual Studio 8, this is accmplished by running:

```
C:\Program Files (x86)\Microsoft Visual
```

To rebuild all EDT applications and libraries, run...

```
nmake
```

To build a specific application, run...

`nmake` *`appname.exe`*

Linux

Use the make utility and gcc developer tools. Depending on your Linux distribution, you may need to first install the development tools to accomplish this. For details, see the documentation for your specific Linux distribution.

In a terminal window, navigate to the installation directory `/opt/EDTpdv`. To rebuild all EDT applications and libraries, run...

```
make
```

To build a specific application, run...

`make` *`appname`*

Note that the vlviewer GUI application is not in the makefile's *all* rules list; therefore to rebuild it you must run...

```
nmake vlviewer
```

(For details on the `vlviewer` framegrabeber GUI, see your EDT framegrabber users guide.)

Initializing the Board

The board is initialized with the `clsiminit` utility, which takes as an argument the path to a camera configuration file.

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.

EDT provides camera-specific files for numerous camera models, and several generic (`generic*cl.cfg`) templates that you can use to create your own files if desired.

Camera configuration files are editable text files. For example, to use `generic8cl.cfg`, edit the file and change the parameters (width and height for example) to match those of the camera you will simulate. Alternatively, if you will simulate a camera for which a file already exists, you can use that file instead; the `clsiminit` application reads the same files as the EDT frame grabber initialization utility: `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 2).

In addition to directives that apply to both frame grabbers and simulators, the simulator also accepts directives that set its pixel clock speed, blanking interval, and other parameters (listed in Simulator-specific Configuration File Directives on page 9). 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`.

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 VisionLink CLS 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`.

clsiminit usage

```
clsiminit --help
```

`-u` *`unit`*	The unit number of the VisionLink CLS (by default, 0).
`-B`	Do not load the FPGA configuration file.
`-q`	Disables program output (quiet mode).
`-s`	Sets the VisionLink 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 serial 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

The provided configuration files are framegrabber-specific; to accurately simulate a camera it is usually necessary to add simulator-specific directives.We recommend making a copy of an existing file then adding the desired directives to your copy. Start by modifying the `camera_` so that the new configuration can be distinguished from the original when viewed in configuration dialogs such as `vlviewer`. Then add any desired simulator-specific directives, as below.

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 can 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 (default 300)
`cls_hcntmax`	total clock cycles per line (default: width / taps + hblank - 1)

One of the next two can 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 (default 400)
`cls_vcntmax`	total lines per frame (default: height + cls_vgap - 1)

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 directive 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 Appendix D: About the Camera Link Standard on page 27.

NOTE When using an EDT frame grabber, this same interleave value is then used for deinterleaving within the frame grabber GUI, `vlviewer`.

Sending Image Data From Host Via DMA

The simple_clsend and send_tiffs demonstration applications show how to use the EDT library to send images (up to 4096 pixels wide by 15,000 lines) through the VisionLink CLS. The programs read a list of images and send each one through the simulator. Although simple_clsend and send_tiffs support only TIF images, the simple_clsend source code includes stubs for other image formats. You can use these stubs to modify the application to support other file formats as desired.

NOTE Before running simple_clsend, you must first initialize the VisionLink CLS with clsiminit. For details, see Software on page 7.

Both applications look in the file imagelist.txt (by default, in the current directory) and read in a list of image filenames, one per line, to send through the VisionLink CLS. simple_clsend 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 any tags specific to send_tiffs. The application then opens the CLS board, and loops through the list of images one at a time, sending each image to that board in sequence.

NOTE simple_clsend is recommended for most cases, because it includes stubs for image formats other than TIF. Use send_tiffs only if you need such rarely used functions as fill, vgap, and hstart.

simple_clsend usage and examples

```
simple_clsend --help
```

-u *unit*	The unit (board) number, default 0.
-c *channel*	The channel (connector) number, default 0
-l *loops*	The number of times to loop through the image list (default 1, 0 to loop indefinitely)
-m	Load all images into memory up front (caution: must have adequate memory!)
-rgb4	Emulate RGB with 4-taps (legacy, shouldn't be needed for VisionLink CLS)
-d *directory*	Images directory - if specified, cycles through all valid files in the given directory
-i *listfile*	Imagelist filename - if specified, rotates through all images referenced in the file (format for imagelist file is one filename [path] per line)
-t *#images*	The number of images to send (must be feweer than the number found in the image list)
-h	This help message

The simplest way to run simple_clsend is to run it with no arguments:

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

This will cycle through all the images in the (default) imagelist.txt file one time, and output the data through the simulator.

To send all the files named in list1.txt 10 times:

```
simple_clsend -l 10 -i list1.txt
```

To loop through all images, using 10 instead of the default 4 ring buffers, run

```
simple_clsend -N 10
```

If imagelist.txt lists, for example, 100 images, of which you wish to send only the first twenty, enter:

```
simple_clsend -t 20
```

send_tiffs usage and examples

```
send_tiffs --help
```

-u *unit*	The unit (board) number, default 0
-c *channel*	The channel (connector) number, 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 (for `send_tiffs` only)
`-B` *`FillB`*	Set the right fill value to FillB (for `send_tiffs` only)

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, enter:

```
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 code overview

Like `simple_clsend`, `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 2).

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.

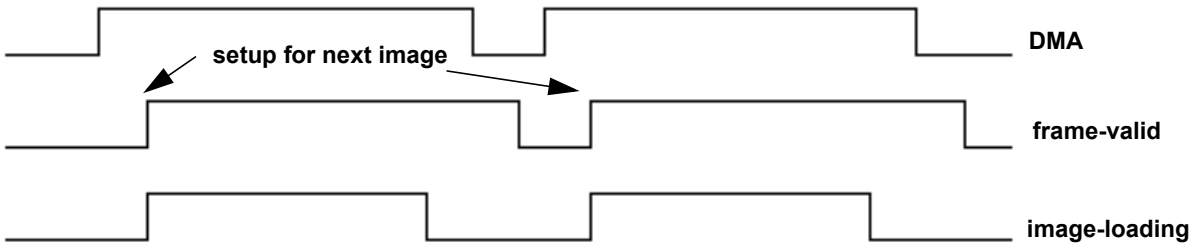
The setup required in step 2 uses the EDT library Event system to ensure that an image is never missed:

```
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 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 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.

Image list format

Each line of an image list file accessed by `simple_clsend` or `send_tiffs` 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.

NOTE `simple_clsend` recognizes only the image filenames and comments; all other directives shown here apply only to `send_tiffs`.

Comments begin with `#`, so whenever `simple_clsend` or `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 VisionLink CLS will send `image4.tiff` with the `fillA, fillB, vgap` and `hstart` values used for `image3.tiff` in the previous line.

Image Data From Internal Counter

As an alternative to sending image data via host DMA, you can use an 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 least 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 frame grabber DMA. Because data does not come from the host, no DMA transfer is involved and no PCIe bandwidth is consumed.

Firmware

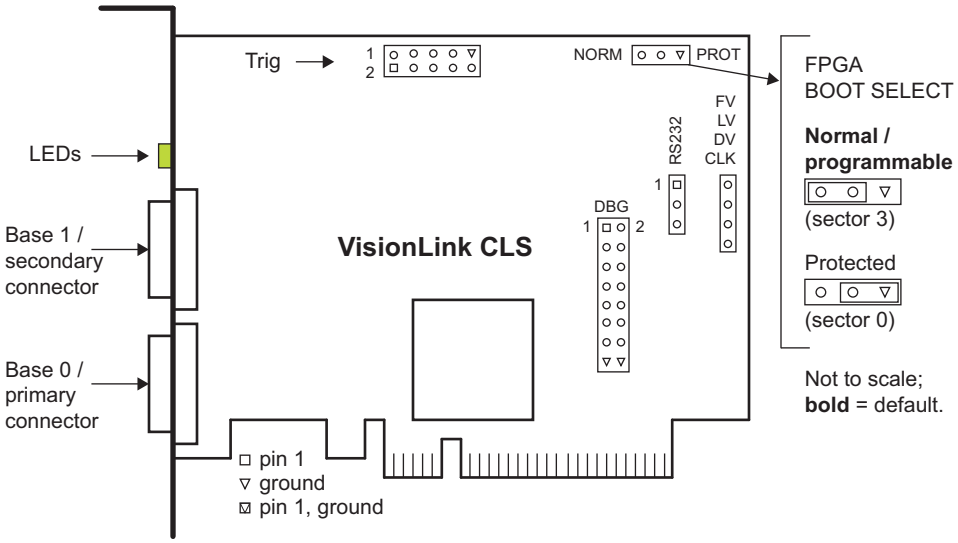
The simulator's operation is controlled by FPGA firmware that loads on boot from an on-board flash PROM. As shipped, each board's flash PROM is pre-loaded with the firmware that was current at the time of production. Updates to this firmware may occasionally be necessary or desired.

The most recent firmware is included in the PDV installation package in the form of a .bit file; in the case of the VisionLink CLS the bitfile will be named `visionlink_cls.bit`. Any firmware updates will be noted in the CHANGELOG_PDV.txt file that is included as part of each PDV software package.

Updating the board's flash PROM from a given .bit file is accomplished using an FPGA update utility. At the time of this writing, the utility for updating the VisionLink CLS is TBD. In the meantime, contact EDT if you have a reason to believe you need to update your firmware.

Appendix A: Board Diagram

Figure 3. VisionLink CLS



Appendix B: Pin Assignments

Table 3 shows the SDR26 pin assignments for the Camera Link connectors.

Camera Link (SDR26) Connectors

The Camera Link (SDR26) connector pin assignments for various operating modes are shown below.

Table 3. Pin assignments – SDR26 connector

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

Appendix C: 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 FPGA to configure the simulated data.

The FPGA configuration file `visionlink_cls.bit` defines these types of registers:

- Camera Link registers;
- simulator registers;
- timing registers

In the descriptions below, the first register address applies to channel 0 and the second applies to channel 1. 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 / 0x80 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 / 0x8B 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 / 0x81 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 / 0x82 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 / 0x85 [Reserved]

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

0x07 / 0x87 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 / 0x8A 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 / 0x8B 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 / 0x8C Serial Data Control) is set and: • FVINT is set, or • TRANSMIT_RDY and EN_TX_INT (in 0x0C / 0x8C Serial Data Control) are both set, or • RECEIVE_RDY and EN_RX_INT (in 0x0C / 0x8C 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 / 0x90 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 / 0x8C 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 / 0xA4 Baud Rate is zero, sets the baud rate: 009600 0119,200 1038,400 11115,200
5	CL_RECEIVE_RDY	Set to clear RECEIVE_RDY in 0x0B / 0x8B 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.

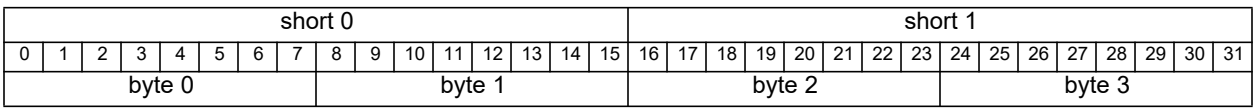
0x0F / 0x8F Utility

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

Accommodates a big-endian host. The 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 4 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 4 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 4. Data Word Structure Without Swapping



0x10 / 0x90 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 / 0xA0 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 / 0xA4 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 / 0x8C Serial Data Control. The baud rate is computed from this value as follows: `(20MHz/(baud_rate*16))-2` For example, 0x80 = 9600 baud.

0x28 / 0xA8 Camera Link Data Path

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

A value of 0x37 configures the VisionLink 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 / 0xC0 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 / 0xCA–CB Vertical Active and 0x4C–4E / 0xCC–CE Vertical Count Maximum were set to infinity). When clear, frame-valid determines which lines 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 line are filled with the values from 0x44 / 0xC4 FillA and 0x46 / 0xC6 FillB respectively, and the positions of the margins are determined by 0x58–59 / 0xD8–D9 Horizontal Read Valid Start and 0x5A–5B / 0x5A–5B Horizontal Read Valid End. When clear, the entire line 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	[no name]	Not used.
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 / 0x82 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 VisionLink 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 / 0xC1 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 / 0xD4–D5 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 / 0xC2 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 line until a trigger is detected. A Dalsa linescan camera starts the next line when it detects a rising edge on the CC1 line.

0x43 / 0xC3 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 / 0xC2 Camera Link Configuration C, when used for line-valid triggering. To any delay, add an additional six pixel clock cycles of pipeline delay.

0x44 / 0xC4 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 line, as determined by 0x58–59 / 0xD8–D9 Horizontal Read Valid Start.

0x46 / 0xC6 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 line, as determined by 0x5A–5B / 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 / 0xC8–C9 Horizontal Count Maximum, after which it clears to zero. Therefore, a value of 0x04FF in 0x48–49 / 0xC8–C9 Horizontal Count Maximum causes `hcnt` to count repeatedly from 0x0000 to 0x04ff, for a line 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 / 0xD4–D5 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 / 0xD6–D7 Horizontal Line Valid End. Therefore, if 0x54–55 / 0xD4–D5 Horizontal Line Valid Start is set to 0x0000 and 0x56–57 / 0xD6–D7 Horizontal Line Valid End to 0x0400, then the line-valid signal is on for 1024 pixel clock cycles.

The registers 0x58–59 / 0xD8–D9 Horizontal Read Valid Start and 0x5A–5B / 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 / 0xC4 FillA and 0x46 / 0xC6 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 pixel clock boundaries, so the number of bytes per line 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 lines that are an even multiple of four pixels long.

Frame-valid starts on the first line of the frame, then remains high until the final line of the frame, as determined by 0x4A–4B / 0xCA–CB Vertical Active. However, frame-valid can start at the beginning of the first line, or some number of pixel clock cycles into the first line; likewise, frame-valid can remain asserted until the very end of the last line, or go low some number of pixel clock cycles before the very end of the last line. The registers 0x50–51 / 0xD0–D1 Horizontal Frame Valid Start and 0x52–53 / 0xD2–D3 Horizontal Frame Valid End determine the position within the first and last line 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 line at the position determined by 0x50–51 / 0xD0–D1 Horizontal Frame Valid Start. During that same line, a 24-bit counter called `vcnt` is zeroed, and then increments at the end of each line. When the 16 least significant bits of `vcnt` are equal to the value in 0x4A–4B / 0xCA–CB Vertical Active, then frame-valid is turned off during that line at the position determined by 0x52–53 / 0xD2–D3 Horizontal Frame Valid End. When `vcnt` is equal to the 24-bit value in 0x4C–4E / 0xCC–CE Vertical Count Maximum, the simulator is ready to start the next frame on the following line (assuming that sufficient DMA data is available). Thus, the value in 0x4C–4E / 0xCC–CE Vertical Count Maximum plus one is the number of lines spent on a given image, including active lines and lines spent in vertical blanking.

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

By default, 0x54–55 / 0xD4–D5 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 / 0xD0–D1 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 / 0xCA–CB Vertical Active.
2. Set the value of 0x50–51 / 0xD0–D1 Horizontal Frame Valid Start to a value between that of 0x56–57 / 0xD6–D7 Horizontal Line Valid End and 0x48–49 / 0xC8–C9 Horizontal Count Maximum, inclusive.

0x48–49 / 0xC8–C9 Horizontal Count Maximum

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

Bit	Name	Description
15–0	[no name]	The line 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 / 0xCA–CB Vertical Active

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

Bit	Name	Description
15–0	[no name]	The number of active lines 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 / 0xCC–CE Vertical Count Maximum

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

Bit	Name	Description
23–0	[no name]	The total number of lines per frame period, minus one. For example, with a value of 0x2FF, there are 768 lines 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 / 0xD0–D1 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 line where the frame-valid signal starts. Frame-valid signal usually stays high for many lines between its start and end. 0x50 contains the least significant bits; 0x51 contains the most significant bits.

0x52–53 / 0xD2–D3 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 line where the frame-valid signal ends. Frame-valid signal usually stays high for many lines between its start and end. 0x52 contains the least significant bits; 0x53 contains the most significant bits.

0x54–55 / 0xD4–D5 Horizontal Line Valid Start

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

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

0x56–57 / 0xD6–D7 Horizontal Line Valid End

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

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

0x58–59 / 0xD8–D9 Horizontal Read Valid Start

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

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

0x5A–5B / 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 line 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 / 0xC0 Camera Link Configuration A is true). If RVEN=0 in 0x40 / 0xC0 Camera Link Configuration A, then both 0x58–59 / 0xD8–D9 Horizontal Read Valid Start and 0x5A–5B / 0x5A–5B Horizontal Read Valid End are ignored and read-from-DMA is true whenever line-valid is true.

Appendix D: 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 line.
Frame-valid	The frame-valid signal goes true at the start of the first line line of the image, and remains true until the end of the final 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 serial 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 serial 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 serial and camera control wires of the second cable are not used.

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

In full mode, the serial 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.

In 80-bit mode, some of the image enable signals are repurposed to send image data, increasing the data payload to 80 bits per pixel clock cycle. This allows for a maximum possible throughput of 850MB/s.

Revision Log

Below is a history of modifications to this guide.

Date	Rev	By	Pp	Detail
20171214	0000	PH,RH, JM,CH	All	• Created new guide.