HUNT ENGINEERING
Chestnut Court, Burton Row,
Brent Knoll, Somerset, TA9 4BP, UK
Tel: (+44) (0)1278 760188,
Fax: (+44) (0)1278 760199,
Email: sales@hunteng.co.uk
www.hunteng.co.uk
www.hunt-dsp.com

Camera Link Area-scan Camera Interface

V1.0 R.Williams 14-04-06

The HERON-FPGA and HERON-IO families are ranges of HERON modules with FPGAs, often combined with some interface capability. The HERON-FPGA family in particular provides an FPGA along with a large number of signals routed to general-purpose connectors. These modules are suitable for connecting to digital cameras, where the control of the camera and image capture can be performed by the FPGA fitted to the module.

For the development of the FPGA function, HUNT ENGINEERING provides a Hardware Interface Layer written in VHDL. This layer allows developers to focus on the application-specific parts of the system. All of the module hardware should be accessed using parts from the library.

Through the Camera Link Area-scan Camera IP HUNT ENGINEERING provides a structured starting point for the development of an area-scan camera interface. The example IP includes several components suitable for processing a stream of camera data. It is intended to be a generic tutorial, so does not address any camera specific issues.

History

Rev 1.0	Developed from previous CameraLink examples

Concepts

FPGA devices are programmable hardware, which can be configured to meet the needs of your system. As with a microprocessor, the function of the FPGA is controlled entirely by the program – and by the peripherals the FPGA is connected to.

A HERON module has access to FIFOs for communicating with other modules in the system – there may be up to 6 input FIFOs and 6 output FIFOs. It may have additional interfaces, such as ADCs, or level-shifting buffers, allowing it to interface to the real world.

The HERON-FPGA family use Xilinx re-configurable logic devices. These can be programmed at low level, but can also be programmed using high level blocks, such as FIR filters, FFT transforms and so on. This process is covered in a separate paper.

The Camera Link Example IP

Camera Link is a communication interface that was specifically developed by camera and frame-grabber manufacturers for use with vision applications. Camera Link works in one of three different configurations, Base, Medium and Full. The Camera Link example has been developed specifically for the Base Configuration only. Using this configuration however, it is possible to support cameras from 8-bit pixel resolutions up to 24-bit RGB. Advanced users could adapt this VHDL to be able to use Medium and Full CameraLink configurations.

Camera Link cameras transmit and receive using the LVDS signal standard. The example IP is designed to connect to a Camera Link camera and must therefore interface via LVDS. When using the example IP you will need to ensure that LVDS is supported by the FPGA module you are using.

The minimum requirement is that LVDS inputs can be supported by the FPGA. This will allow a stream of camera data to be received and processed by the FPGA.

If LVDS outputs are also possible with the FPGA then Camera Control outputs can be added. With LVDS outputs it also becomes possible to implement the bi-directional RS232 link of the Camera Link standard.

Please refer to the IO capabilities of the FPGA module you are using to determine which features of CameraLink are available.

The Camera Link IP is supplied as a set of VHDL source files along with instructions on its use in the remainder of this document. The VHDL includes components to interface to Camera Link, components to control the flow of camera data through the FPGA and out to a HERON FIFO and components to implement an RS232 connection with the camera.

The part of the design tree that includes the files CamLink_Hsb.vhd and below contains a set of registers that are programmed over the HSB message interface in order to control camera interface operation. The HSB interface also provides a mechanism for transmitting and receiving RS-232 messages between the Camera Link camera and the FPGA camera interface.

The part of the design tree that includes the file CamLink_IF.vhd and below contains the Camera Link interface. The file Camera.vhd contains generic camera processing functions. All of these files can be modified to make your own camera interface.

This tutorial assumes that you are using the Xilinx ISE design tools to build the Camera Link example.

What the IP Does

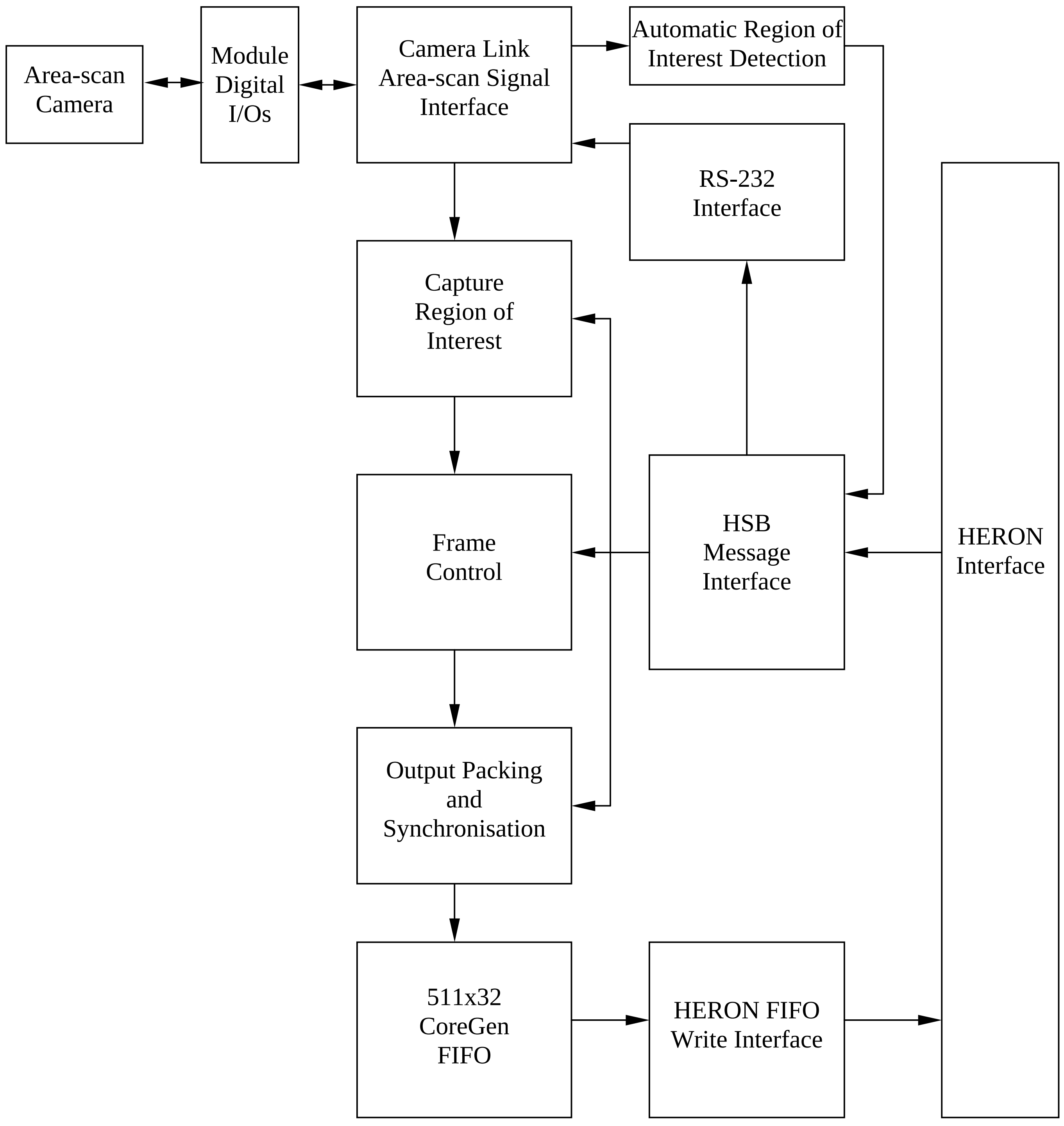
The Camera Link example IP provided on the HUNT ENGINEERING CD implements a generic area-scan camera interface, with automatic frame-size and region of interest detection and programmable region of interest for capture, and frame capture control.

VHDL is provided for connecting to a camera that operates using Camera Link. There are two build versions of the VHDL. One build option is for cameras operating at 24MHz and below and the other option for cameras working at frequencies above 24MHz.

The VHDL provided in the example IP includes source code for two different frequency Camera Link de-serializers. Each de-serializer design works with a different range of camera pixel clock frequency. The appropriate frequency range de-serializer must be used based on the pixel clock frequency of the camera you are using.

For cameras with pixel clocks of 24MHz or below the `USE_LOW_FREQ_DESERIALISER` Boolean must be set to `TRUE` before the example is built, and for cameras with pixel clock of 24MHz and above the Boolean must be set to `FALSE`.

Functional Block Diagram


Area-scan Camera
Module Digital I/Os
Camera Link Area-scan Signal Interface
Automatic Region of Interest Detection
RS-232 Interface
Capture Region of Interest
HSB Message Interface
HERON Interface
Frame Control
Output Packing and Synchronisation
511x32 CoreGen FIFO
HERON FIFO Write Interface

Camera Inputs

The Camera Link interface transmits camera control and data signals using the LVDS signalling standard. In order to interface to LVDS signals the correct input and output buffer components must be used in the FPGA design. This is done inside the file 'top.vhd'.

The example IP requires as inputs one LVDS camera clock and four LVDS camera data lines. The example IP also drives as outputs four LVDS camera control signals, receives one RS232 input from the camera and transmits one RS232 output to the camera. For FPGAs that support LVDS input buffers only, it is possible to omit the Camera Control outputs and RS232 connections.

The LVDS standard is differential. Therefore, one LVDS connection will involve a positive signal and a negative signal complement. A single LVDS connection requires the use of a P and N signal pair in the Xilinx design. The Digital I/O pin assignment for the Camera Link example must ensure that P and N pairs are correctly created in the FPGA design.

The remainder of this document assumes that the Camera Link camera has been connected as shown in the table below using the first two Digital I/O connectors of an FPGA module. Please refer to the example IP files for examples of other schemes that can be used for the various module types.

Connector A/B	Camera Link Signal
CONN_A0	X0–
CONN_A1	X0+
CONN_A2	X1–
CONN_A3	X1+
CONN_A4	X2–
CONN_A5	X2+
CONN_A6	Xclk–
CONN_A7	Xclk+
CONN_A8	X3–
CONN_A9	X3+
CONN_B0	SerTC–
CONN_B1	SerTc+
CONN_B2	SerTFG–
CONN_B3	SerTFG+
CONN_B4	CC1–
CONN_B5	CC1+
CONN_B6	CC2–
CONN_B7	CC2+
CONN_B8	CC3–
CONN_B9	CC3+
CONN_B10	CC4–
CONN_B11	CC4+

Signal Voltage Levels

The Camera Link standard uses Low Voltage Differential Signalling (LVDS) for the electrical interface layer. When using the Camera Link example IP please check that the FPGA module you are using is capable of connecting to LVDS.

For LVDS inputs to the FPGA such as the Camera Link clock and data signals you must ensure that LVDS input buffers are available with your FPGA.

If you intend to provide output connections to the camera such as the Camera Link control signals or bi-directional RS232 connection you must also ensure that LVDS output buffers are supported. If using output buffers you will need to ensure that the necessary Vcco voltage level is available on your module for LVDS and that the LVDS standard is supported.

Differential Termination

Differential signalling standards such as LVDS need to be terminated with a resistor between the two halves of the differential pair. For LVDS, a 100-Ohm termination resistance is required for every signal pair received by the FPGA.

Early Xilinx families provide this termination on-chip, by terminating each half of the signal in a split termination (one to power and the other to ground). While this offers the correct equivalent termination as far as the AC signal is concerned, it also adds significantly to the static power supply current. For this reason HUNT ENGINEERING do not recommend using this method.

On boards that have FPGAs using this type of termination HUNT ENGINEERING provide, external to the FPGA, true differential termination resistors. For the Camera Link example, it is therefore necessary for 100-Ohm resistor packs to be soldered to the board for signals that will be used as differential inputs. Please note that resistor packs must not be added for the differential LVDS outputs.

Later families of FPGAs implement a true differential termination that can be selected in the FPGA design. In this case HUNT ENGINEERING do not provide the differential termination resistors external to the FPGA.

To use this termination on a differential termination capable FPGA, the `DIFF_TERM` attribute must be set in the design. An example of this is included in the section on building the example IP.

Ordering External Termination

If you intend to use an FPGA that requires the external 100-Ohm termination for LVDS, please ensure that the fitting of resistor packs is requested when you order your FPGA module.

Assuming the same camera signal pin-out as shown in the earlier section 'Camera Inputs' then 100-Ohm termination resistor packs will be needed for all sites on Connector A along with a single resistor pack for the SerTFG signal pair on Connector B.

If when you ordered your HERON-FPGA module you did not inform HUNT ENGINEERING that you would be interfacing to Camera Link cameras, then your board will not have these resistors fitted. In this case you can either contact HUNT ENGINEERING to discuss having the resistors fitted at the factory, or alternatively you can fit the resistors yourself. Please note, although HUNT ENGINEERING is happy for you to fit the resistors yourself, any damage done to the board in doing so is not covered under the warranty.

Camera Speeds

The example IP supplied by HUNT ENGINEERING has been specified for operation using a Camera pixel clock frequency of 50MHz or less. There are two build versions of the IP, one using a Low Frequency de-serializer for pixel clocks up to 24MHz and one using a High Frequency de-serializer for pixel clocks above 24MHz.

The Camera Link standard uses 'Channel Link' for the physical layer. Channel Link consists of a driver and receiver pair. The driver accepts 28 bits of data and a clock. The data is serialised 7:1 into four data streams and transmitted with a dedicated clock over 5 LVDS pairs. The receiver accepts the four LVDS data streams and clock and then recreates the original 28-bit data stream along with the clock.

For those users who have cameras with pixel clocks above 50MHz the User Constraints File for the project needs to be modified to reflect the particular operating frequency. When this has been done the bitstream must be regenerated and the place and route report file checked to see those time-specifications were met.

Automatic Region of Interest Detection

The module AUTO_ROI performs automatic region of interest detection. The module allows the frame size and active areas of the image to be automatically detected and read over the HSB message interface.

This information, if used, enables the controlling software to automatically calculate the window for the capture region of interest process by using values that directly relate to the observed operation of the camera.

Capture Region of Interest

The module ROI performs a region of interest operation by using a pixel counter and a line counter, along with a pixel start position, pixel stop position, line start position and line stop position.

The pixel start, pixel stop, line start and line stop values are all programmed over the HSB message interface. Each of these values can be pre-calculated using the Automatic Region of Interest logic, or they can be set directly from values supplied by the user.

Frame Control

After the capture region of interest has processed each frame, whether that frame will be output or not depends on the frame-control component. The continuous mode and N-frames mode are programmed over the HSB message interface.

Output Packing and Synchronisation

One word, equal to 0 is added at the end of each camera line, apart from the last line of a frame, where one word with all bits set high (the value FFFFFFFFh) is added to indicate end of frame. In order to avoid the synchronisation values being present in the data, the bottom byte of camera data is modified to prevent the values 00h and FFh (a value of 00h will become 01h and a value of FFh will become FEh).

CoreGen FIFO and HERON FIFO Interface

After the data has been processed by the output packing and synchronisation component, the data is fed into a Core-generator generated FIFO. The FIFO is a 32-bit word, 511 word deep asynchronous FIFO built using Block RAM.

When there is any data in the CoreGen FIFO, it is read out and placed in the HERON FIFO Write Interface. The FIFO number used for output is programmed over the HSB message interface. If the HERON FIFO becomes full, eventually the CoreGen FIFO will become full and camera data will be lost. If the CoreGen FIFO becomes full, LED0 will become illuminated.

RS-232 Interface

Camera Link provides two LVDS pairs for serial communication between the 'frame grabber' and camera. The first LVDS pair 'SerTC+/SerTC–' is provided for communication To the Camera (TC) and the second pair 'SerTFG+/SerTFG–' is provided for communication To the Frame-Grabber (TFG).

The RS-232 Interface is connected to HSB. This enables HSB messages to be used to send RS-232 commands to the camera and receive camera status information back. The RS-232 Interface uses a 16-byte transmit buffer and a 16-byte receive buffer. These buffers allow RS-232 transmission to be decoupled from HSB.

The Baud rate of the RS-232 connection can be programmed over HSB.

HSB Control Registers

The following table defines the control registers that can be set by sending messages to the FPGA using HSB.

HSB Address Byte (decimal)	Register Function	Description
0	Pixel Start 0 Register	LSB of pixel start register for Capture Region Of Interest (register is 12bits)
1	Pixel Start 1 Register	Top 4 bits of pixel start register for Capture Region Of Interest (register is 12bits)
2	Pixel Stop 0 Register	LSB of pixel stop register for Capture Region Of Interest (register is 12bits)
3	Pixel Stop 1 Register	Top 4 bits of pixel stop register for Capture Region Of Interest (register is 12bits)
4	Line Start 0 Register	LSB of line start register for Capture Region Of Interest (register is 12bits)
5	Line Start 1 Register	Top 4 bits of line start register for Capture Region Of Interest (register is 12bits)
6	Line Stop 0 Register	LSB of line stop register for Capture Region Of Interest (register is 12bits)
7	Line Stop 1 Register	Top 4 bits of line stop register for Capture Region Of Interest (register is 12bits)
8	Frame Control Register	Set number of frames to capture and send (range 1 to 15, value set in bottom 4-bits) Or, set bit 4 for continuous capture (i.e. write 0x10)
9	Camera Control Register	Bit 0 is used to directly set Camera Link Camera Control 1 (CC1). Bit 1 is used to directly set Camera Link Camera Control 2 (CC2). Bit 2 is used to directly set Camera Link Camera Control 3 (CC3). Bit 3 is used to directly set Camera Link Camera Control 4 (CC4). Bit 4 is used to control the ODD-ONLY function. When this bit is set high only odd pixels are output in each line, and only odd lines are output in each frame. Bit 5 is used to set the polarity of the Camera Link control signals, FVAL, LVAL and DVAL. Set to 0 if all signals are active low. Set to 1 if all signals are active high.

HSB Address Byte (decimal)	Register Function	Description
9	Camera Control Register	Bit 6 is used to indicate whether the connected camera provides a Camera Link DVAL signal. Set to 0 if no DVAL signal is provided. Set to 1 if a DVAL signal is provided. Bit 7 is used to reset the Auto-Region-of-Interest logic. Set to 1 to assert reset and then set to 0 to de-assert reset.
10	FIFO Number Register	One Hot setting for the HERON Output FIFO number that the data will be sent on.
11	Camera Mode Register	Bits 0 to 3 MUST always be written with the value 4. Bits 4 and 5 are used to control a data shift which is required when using 10 or 12 bit Camera Link cameras, as follows: Bit 5 Bit4 Function Data Shift 0 0 8-bit camera none 0 1 10-bit camera right x 2 1 0 12-bit camera right x 4 1 1 14-bit camera right x 6
12	Baud Rate Register	Bottom 4 bits of this register are used to set Baud rate of the RS-232 interface as follows: 0000 4800 0001 9600 0010 19200 0011 38400 0100 115200
13	Transmit Data Register	A write to this register will place the data in the next free position of the RS-232 transmit buffer.

The following table defines the status registers that can be read from the FPGA using HSB.

HSB Address Byte (decimal)	Register Function	Description
0	First Pixel 0 Register	Returns the bottom byte of the 'First-Pixel' value generated by the Auto-Region-of-Interest logic.
1	First Pixel 1 Register	Bits 0 to 3 return the top 4-bits of the 'First-Pixel' value generated by the Auto-Region-of-Interest logic. Bits 4 to 6 are undefined. Bit 7 returns the state of the Auto-Region-of-Interest Logic. 0 indicates that the Auto-ROI values cannot be used. 1 indicates the Auto-ROI values are valid and safe to use.
2	Last Pixel 0 Register	Returns the bottom byte of the 'Last-Pixel' value generated by the Auto-Region-of-Interest logic.
3	Last Pixel 1 Register	Bits 0 to 3 return the top 4-bits of the 'Last-Pixel' value generated by the Auto-Region-of-Interest logic. Bits 4 to 7 are undefined.
4	First Line 0 Register	Returns the bottom byte of the 'First-Line' value generated by the Auto-Region-of-Interest logic.
5	First Line 1 Register	Bits 0 to 3 return the top 4-bits of the 'First-Line' value generated by the Auto-Region-of-Interest logic. Bits 4 to 7 indicate bit activity in the 24-bit 'Base Configuration' Camera Link data (from bit 23 down to bit 0), as follows: Bit set high — Indicates 4 — Activity in bit 9 5 — Activity in bit 11 6 — Activity in bit 13 7 — Activity in bit 15
6	Last Line 0 Register	Returns the bottom byte of the 'Last-Line' value generated by the Auto-Region-of-Interest logic.
7	Last Line 1 Register	Bits 0 to 3 return the top 4-bits of the 'Last-Line' value generated by the Auto-Region-of-Interest logic. Bits 4 to 7 are undefined.
13	Transmit Data and Status Register	A read from this register returns the number of free spaces in the transmit buffer.

HSB Address Byte (decimal)	Register Function	Description
14	Receive Data Register	A read from this register returns the next data byte available in the RS-232 receive buffer.
15	Receive Status Register	A read from this register returns the number of data bytes in the RS-232 receive buffer.
16	Pixel Total 0 Register	Returns the bottom byte of the 'Pixel-Total' value generated by the Auto-Region-of-Interest logic.
17	Pixel Total 1 Register	Bits 0 to 3 return the top 4-bits of the 'Pixel-Total' value generated by the Auto-Region-of-Interest logic. Bits 4 to 7 return zero.
18	Line Total 0 Register	Returns the bottom byte of the 'Line-Total' value generated by the Auto-Region-of-Interest logic.
19	Line Total 1 Register	Bits 0 to 3 return the top 4-bits of the 'Line-Total' value generated by the Auto-Region-of-Interest logic. Bits 4 to 7 return zero.

Building the Example IP

This tutorial assumes that you are using the Xilinx ISE design tools to build the Camera Link example.

In order to build the Camera Link Example IP it is necessary to start from an existing ISE project. This tutorial assumes that the starting point will be the Getting Started example (Example1) for the module type you are using.

It is expected that before working through the Camera Link example you have already worked through and understood the Getting Started example for your module.

Copying Examples from the HUNT ENGINEERING CD

On the HUNT ENGINEERING CD, under the directory "fpga" you can find directories for each module type.

There are two ways that you can copy the files from the CD. The directory tree with the VHDL sources, bit-streams etc can be copied directly from the CD to the directory of your choice. In this case there is no need to copy the .zip file, but the files will be copied to your hard drive with the same read only attribute that they have on the CD. In this case all files in the example directory will need to be changed to have read/write permissions. It is a good idea to leave the permissions of the "Common" directory set to read only to prevent the accidental modification of these files.

To make the process more convenient we have provided a zip file, which is a zipped image of the same tree you can see on the CD. If you "unzip" this archive to a directory of your choice, you will have the file permissions already set correctly.

Creating a Camera Link Project

To make a project that uses the Camera Link example IP, follow the steps below:

1. Make a new directory named '`CamLink_Cam`'.
2. Copy the '`Example1`' and '`Common`' directories for your module type from the HUNT ENGINEERING CD into the new directory.
3. Modify the file attributes of the file '`CamLink_Cam\Common\top.vhd`' to be read/write.
4. Rename the '`Example1`' directory to '`CamLink_Ex`'.
5. In the '`ISE`' sub-directory rename the ISE project to '`CamLink_Ex.ise`' and rename the User Constraints File to '`CamLink_Ex.ucf`'.
6. Launch the ISE Project Navigator and then open the new project '`CamLink_Ex.ise`'. As the project is opened there will be an information box saying that the previous UCF file does not exist. Simply click OK to remove the box.
7. Using the menu item '`Project->Add Source`…' add the '`CamLink_Ex.ucf`' file to the project and associate it with the design unit '`top`'.

8. Open the '`User_Ap1.vhd`' source file in the ISE text editor. A component declaration must be added to include the top level of the Camera Link example IP. Add the following VHDL immediately after the `architecture` statement:

```
component CAMLINK_EX
  generic (
    Use_Low_Freq_Deserialiser : boolean := TRUE
  );
  port (
    -- Clock and Reset
    RESET         : in  std_logic;
    CLOCK         : in  std_logic;
    -- CameraLink interface
    CAM_CLK       : in  std_logic;
    CAM_DATA      : in  std_logic_vector(3 downto 0);
    CAM_CTRL      : out std_logic_vector(3 downto 0);
    CAM_SER_TC    : out std_logic;
    CAM_SER_TFG   : in  std_logic;
    -- HERON Output FIFO interface
    FCLK_G        : in  std_logic;
    OUTFIFO_READY : in  std_logic_vector(5 downto 0);
    OUTFIFO_WRITE : out std_logic_vector(5 downto 0);
    OUTFIFO_D     : out std_logic_vector(31 downto 0);
    -- FIFO Full LED
    FIFO_FULL_LED : out std_logic;
    -- User Message interface
    MSG_CLK       : out std_logic;
    MSG_DIN       : in  std_logic_vector(7 downto 0);
    MSG_ADDR      : in  std_logic_vector(7 downto 0);
    MSG_WEN       : in  std_logic;
    MSG_REN       : in  std_logic;
    MSG_DONE      : in  std_logic;
    MSG_COUNT     : in  std_logic;
    MSG_CLEAR     : in  std_logic;
    MSG_READY     : out std_logic;
    MSG_SEND_MSG  : out std_logic;
    MSG_CE        : out std_logic;
    MSG_DOUT      : out std_logic_vector(7 downto 0);
    MSG_SEND_ID   : out std_logic;
    MSG_LAST_BYTE : out std_logic
  );
end component;
```

9. In the ‘`User_Ap1.vhd`’ source file add a component instantiation for the ‘`CAMLINK_EX`’ component. Add the following VHDL after the `begin` statement:

```
------------------------------------------------------------
-- Camera Link Example IP component

iCAMLINK_EX : CAMLINK_EX
  generic map (
    Use_Low_Freq_Deserialiser => USE_LOW_FREQ_DESERIALISER
  )
  port map (
    -- Clock and Reset
    RESET         => RESET,
    CLOCK         => OSC3,
    -- CameraLink interface
    CAM_CLK       => CAM_CLK,
    CAM_DATA      => CAM_DATA,
    CAM_CTRL      => CAM_CTRL,
    CAM_SER_TC    => CAM_SER_TC,
    CAM_SER_TFG   => CAM_SER_TFG,
    -- HERON Output FIFO interface
    FCLK_G        => FCLK_G,
    OUTFIFO_READY => OUTFIFO_READY,
    OUTFIFO_WRITE => OUTFIFO_WRITE,
    OUTFIFO_D     => OUTFIFO_D,
    -- FIFO Full LED
    FIFO_FULL_LED => FIFO_FULL_LED,
    -- User Message interface
    MSG_CLK       => MSG_CLK,
    MSG_DIN       => MSG_DIN,
    MSG_ADDR      => MSG_ADDR,
    MSG_WEN       => MSG_WEN,
    MSG_REN       => MSG_REN,
    MSG_DONE      => MSG_DONE,
    MSG_COUNT     => MSG_COUNT,
    MSG_CLEAR     => MSG_CLEAR,
    MSG_READY     => MSG_READY,
    MSG_SEND_MSG  => MSG_SEND_MSG,
    MSG_CE        => MSG_CE,
    MSG_DOUT      => MSG_DOUT,
    MSG_SEND_ID   => MSG_SEND_ID,
    MSG_LAST_BYTE => MSG_LAST_BYTE );
```

10. In the 'fpga' directory of the HUNT ENGINEERING CD there is a sub-directory named 'generic_examples' and below this directory is a sub-directory named 'CameraLink' that contains the VHDL for the Camera Link example IP. Copy the contents of the 'Src' directory from the Camera Link example on the HUNT ENGINEERING CD to the 'Src' directory of the new project.
11. Copy the contents of the 'ISE' directory from the Camera Link example on the CD to the 'ISE' directory of the new project.
12. In the ISE Project Navigator add the following VHDL files from the project's 'Src' directory:

```
CamLink_Ex.vhd
CamLink_IF.vhd
CamLink_Hsb.vhd
Camera.vhd
RS232_Uart.vhd
```

13. Add the Core Generator FIFO source file 'fifo511x32.xco' to the project from the 'ISE' directory.
14. In the CONFIG package of the User-Ap file add the following VHDL:

```
-- To use the low frequency De-Serialiser logic, set TRUE
-- The low frequency De-Serialiser will work with pixel clocks below 24MHz.
-- To use the high frequency De-Serialiser logic, set FALSE
-- The high frequency De-Serialiser will work with pixel clocks above 24MHz.
constant USE_LOW_FREQ_DESERIALISER : boolean := TRUE;
```

15. In the User-Ap file, delete the existing VHDL for Example1 that instantiates the HSB1 component.
16. In the User-Ap file, delete the existing VHDL for passing FIFO data from the HERON Input FIFOs to the HERON Output FIFOs.
17. In the User-Ap file, add the following VHDL between the `architecture` and `begin` statements to declare a new signal:

```
signal FIFO_FULL_LED : std_logic;
```

18. Add the following VHDL at the end of the User-Ap source file to connect the FIFO full signal of the Camera example IP to LED(0):

```
------------------------------------------------------------
-- COREGEN FIFO Full LED. Illuminated when FIFO is full

LED(0) <= FIFO_FULL_LED;

------------------------------------------------------------
-- Drive other LEDs high (inactive)

LED(3 downto 1) <= (others=>'1');
```

Modifying TOP and the User-Ap Entity

The Example VHDL supplied by HUNT ENGINEERING for each module type includes a top level design file named '`top.vhd`', a Hardware Interface Layer for interfacing to external components such as analogue-to-digital devices, HERON FIFOs and SDRAM, and a User-Ap entity within which the user design is implemented.

It is intended that for normal operation the Hardware Interface Layer and top level design file are never modified. However, in the case of the Camera Link Example IP it is necessary to modify '`top.vhd`' and the User-Ap entity definition.

This modification is necessary because the default setting for the Digital I/Os is to use the Xilinx default I/O standard of single-ended Low-Voltage TTL (LVTTL). The Camera Link example IP requires LVDS signalling and so therefore top must be modified to use this I/O standard.

This section defines how this change must be made.

1. The default I/O buffers used in '`top.vhd`' must be replaced with the components, `IBUFDS`, `IBUFGDS` and `OBUFDS`. To do this, firstly declare these components by adding the following VHDL immediately after the architecture statement in '`top.vhd`':

```
component IBUFDS
  port(
    I  : in  std_logic;
    IB : in  std_logic;
    O  : out std_logic );
end component;

component OBUFDS
  port(
    I  : in  std_logic;
    O  : out std_logic;
    OB : out std_logic );
end component;

component IBUFGDS
  port(
    I  : in  std_logic;
    IB : in  std_logic;
    O  : out std_logic );
end component;
```

2. Having declared buffers for differential signalling, add the following VHDL after the `begin` statement:

```
------------------------------------------------------------
-- Camera Link connections

-- Camera Link Clock
lvds_ibufg  : IBUFGDS port map (I=>dio_a(7), IB=>dio_a(6), O=>CAM_CLK);

-- Camera Link Data
lvds_ibuf0  : IBUFDS port map (I=>dio_a(1), IB=>dio_a(0), O=>CAM_DATA(0));
lvds_ibuf1  : IBUFDS port map (I=>dio_a(3), IB=>dio_a(2), O=>CAM_DATA(1));
lvds_ibuf2  : IBUFDS port map (I=>dio_a(5), IB=>dio_a(4), O=>CAM_DATA(2));
lvds_ibuf3  : IBUFDS port map (I=>dio_a(9), IB=>dio_a(8), O=>CAM_DATA(3));

-- Camera Link Control
lvds_obuf0  : OBUFDS port map (I=>CAM_CTRL(0), O=>dio_b(5),  OB=>dio_b(4));
lvds_obuf1  : OBUFDS port map (I=>CAM_CTRL(1), O=>dio_b(7),  OB=>dio_b(6));
lvds_obuf2  : OBUFDS port map (I=>CAM_CTRL(2), O=>dio_b(9),  OB=>dio_b(8));
lvds_obuf3  : OBUFDS port map (I=>CAM_CTRL(3), O=>dio_b(11), OB=>dio_b(10));

-- Camera Link RS232 connection from HERON-FPGAx to camera
lvds_obuf4  : OBUFDS port map (I=>CAM_SER_TC, O=>dio_b(1), OB=>dio_b(0));

-- Camera Link RS232 connection from camera to HERON-FPGAx
lvds_ibuf4  : IBUFDS port map (I=>dio_b(3), IB=>dio_b(2), O=>CAM_SER_TFG);
```

3. Having added the I/O buffer instantiations the previous buffer instantiations need to be deleted for `dio_a(0)` to `dio_a(9)` and `dio_b(0)` to `dio_b(11)`. If the file 'top.vhd' contains any attributes on the existing I/O buffer instantiations then these must be deleted too.

4. Add buffer attributes to '`top.vhd`' for the new buffer instantiations. The following VHDL shows how this would be done if using 3.3V LVDS. If you intend to use LVDS at 2.5V you will need to change the `IOSTANDARD` to `"LVDS_25"`.

   The I/O standards that are possible will depend on the Vcco options and type of FPGA on the module you are using. Please refer to the User Manual for your FPGA module if you are unsure which I/O standards are available.

   ```
   -- Camera Link Buffer Definitions
   attribute IOSTANDARD of lvds_ibufg : label is "LVDS_33";

   attribute IOSTANDARD of lvds_ibuf0 : label is "LVDS_33";
   attribute IOSTANDARD of lvds_ibuf1 : label is "LVDS_33";
   attribute IOSTANDARD of lvds_ibuf2 : label is "LVDS_33";
   attribute IOSTANDARD of lvds_ibuf3 : label is "LVDS_3";

   attribute IOSTANDARD of lvds_obuf0 : label is "LVDS_33";
   attribute IOSTANDARD of lvds_obuf1 : label is "LVDS_33";
   attribute IOSTANDARD of lvds_obuf2 : label is "LVDS_33";
   attribute IOSTANDARD of lvds_obuf3 : label is "LVDS_33";

   attribute IOSTANDARD of lvds_obuf4 : label is "LVDS_33";
   attribute IOSTANDARD of lvds_ibuf4 : label is "LVDS_33";
   ```

5. Check that the file 'top.vhd' contains a declaration of the `IOSTANDARD` attribute shown below. If it does not exist, add it to the file between the `architecture` and `begin` statements.

   ```
   attribute IOSTANDARD : string;
   ```

6. Having changed the number of Digital I/O signals used as general purpose I/Os and having created new Camera Link signals, the User-Ap entity port list must be changed. The following changes should be made to the `USER_AP` component declaration in the file '`top.vhd`':

   Delete the following lines (N represents the connector width, -1 for the module you are using):

   ```
     CONN_A_IN        : in  std_logic_vector(N downto 0);
     CONN_A_OUT       : out std_logic_vector(N downto 0);
     CONN_A_EN        : out std_logic_vector(N downto 0);
     CONN_B_IN        : in  std_logic_vector(N downto 0);
     CONN_B_OUT       : out std_logic_vector(N downto 0);
     CONN_B_EN        : out std_logic_vector(N downto 0);
   ```

Replace with:

```
CAM_CLK           : in  std_logic;
CAM_DATA          : in  std_logic_vector(3  downto 0);
CAM_CTRL          : out std_logic_vector(3  downto 0);
CAM_SER_TC        : out std_logic;
CAM_SER_TFG       : in  std_logic;
CONN_A_IN         : in  std_logic_vector(N downto 10);
CONN_A_OUT        : out std_logic_vector(N downto 10);
CONN_A_EN         : out std_logic_vector(N downto 10);
CONN_B_IN         : in  std_logic_vector(N downto 12);
CONN_B_OUT        : out std_logic_vector(N downto 12);
CONN_B_EN         : out std_logic_vector(N downto 12);
```

7. Similarly change the USER_AP entity instantiation at the end of 'top.vhd' to include the following ports:

```
--
CAM_CLK           => CAM_CLK,
CAM_DATA          => CAM_DATA,
CAM_CTRL          => CAM_CTRL,
CAM_SER_TC        => CAM_SER_TC,
CAM_SER_TFG       => CAM_SER_TFG,
--
```

8. Next change the signal declarations in '`top.vhd`':

   Delete the following lines (N represents the connector width, -1 for the module you are using):

```
signal CONN_A_IN         : std_logic_vector(N downto 0);
signal CONN_A_OUT        : std_logic_vector(N downto 0);
signal CONN_A_EN         : std_logic_vector(N downto 0);
signal CONN_B_IN         : std_logic_vector(N downto 0);
signal CONN_B_OUT        : std_logic_vector(N downto 0);
signal CONN_B_EN         : std_logic_vector(N downto 0);
```

Replace with:

```
signal CAM_CLK           : std_logic;
signal CAM_DATA          : std_logic_vector(3 downto 0);
signal CAM_CTRL          : std_logic_vector(3 downto 0);
signal CAM_SER_TC        : std_logic;
signal CAM_SER_TFG       : std_logic;
signal CONN_A_IN         : std_logic_vector(N downto 10);
signal CONN_A_OUT        : std_logic_vector(N downto 10);
signal CONN_A_EN         : std_logic_vector(N downto 10);
signal CONN_B_IN         : std_logic_vector(N downto 12);
signal CONN_B_OUT        : std_logic_vector(N downto 12);
signal CONN_B_EN         : std_logic_vector(N downto 12);
```

9. Save the changes to the file 'top.vhd'. Repeat the entity changes to USER_AP in the file 'User_Ap1.vhd' as follows:

   Delete the following lines (N represents the connector width, -1 for the module you are using):

```
CONN_A_IN          : in  std_logic_vector(N downto 0);
CONN_A_OUT         : out std_logic_vector(N downto 0);
CONN_A_EN          : out std_logic_vector(N downto 0);
CONN_B_IN          : in  std_logic_vector(N downto 0);
CONN_B_OUT         : out std_logic_vector(N downto 0);
CONN_B_EN          : out std_logic_vector(N downto 0);
```

   Replace with:

```
CAM_CLK            : in  std_logic;
CAM_DATA           : in  std_logic_vector(3  downto 0);
CAM_CTRL           : out std_logic_vector(3  downto 0);
CAM_SER_TC         : out std_logic;
CAM_SER_TFG        : in  std_logic;
CONN_A_IN          : in  std_logic_vector(N downto 10);
CONN_A_OUT         : out std_logic_vector(N downto 10);
CONN_A_EN          : out std_logic_vector(N downto 10);
CONN_B_IN          : in  std_logic_vector(N downto 12);
CONN_B_OUT         : out std_logic_vector(N downto 12);
CONN_B_EN          : out std_logic_vector(N downto 12);
```

10. If you are using a Xilinx FPGA that internally provides the 100-Ohm differential termination across the input differential receiver terminals you will need to specify this in '`top.vhd`' using the `DIFF_TERM` attribute. Place the following VHDL after between the `architecture` and `begin` statements:

```
attribute DIFF_TERM  : string;

attribute DIFF_TERM  of lvds_ibufg : label is "TRUE";

attribute DIFF_TERM  of lvds_ibuf0 : label is "TRUE";
attribute DIFF_TERM  of lvds_ibuf1 : label is "TRUE";
attribute DIFF_TERM  of lvds_ibuf2 : label is "TRUE";
attribute DIFF_TERM  of lvds_ibuf3 : label is "TRUE";
```

Modifying User Constraints

The Camera Link example IP requires timing constraints to be applied to the project to control the frequency of the HSB clock and the frequency of the Camera Link interface logic.

Open the 'CamLink_Ex.ucf' User Constraints File in the 'ISE' directory of the new project, using the ISE text editor. Add the lines shown below:

```
# ========================================
# This constraint must match the freq
# of the HSB clock signal
# ========================================
NET "iuser_ap/iCAMLINK_EX/HSB_CLK" TNM_NET = "HSB_CLOCK";
TIMESPEC "TS_HSB_CLOCK" = PERIOD "HSB_CLOCK" 50 MHz HIGH 50 %;

# ========================================
#
# All of the following Timespecs are for
# camera operation up to pixel clocks of 50MHz
#
# ========================================
NET "iuser_ap/iCAMLINK_EX/iCLINK/rxclk1" TNM_NET = FFS(*) "rxclk1";
NET "iuser_ap/iCAMLINK_EX/iCLINK/rxclk35" TNM_NET = FFS(*) "rxclk35";
TIMEGRP RXCLK_rising  = rising  "rxclk35";
TIMEGRP RXCLK_falling = falling "rxclk35";
TIMESPEC "ts20" = PERIOD "rxclk1" 50 MHz HIGH 50 %;
TIMESPEC "ts20" = PERIOD "rxclk35" 175 MHz HIGH 50 %;
TIMESPEC "ts21" = from RXCLK_falling to RXCLK_rising = 350 MHz;
TIMESPEC "ts22" = from rxclk35    to rxclk1 = 175 MHz;
```

Please note: the above example assumes a project set to use the forward slash character ('/') as the hierarchy separator. If your project is set to use the underscore character ('_') you will need to replace the forward slash with an underscore in the UCF file.

Creating a Bitstream with the Camera Example IP

Before a bitstream can be built a camera link project must be assembled as detailed in the previous sections of this document. Once this has been done, the only remaining step is to check the settings of the CONFIG package.

At the top of the '`User_ap1.vhd`' file there are global settings contained in the CONFIG package that you can use to affect your design (in this case the camera example). The idea is that settings that are often changed are found here. If you have worked through the Getting Started example for your module type you will already be familiar with the settings in the CONFIG package.

The only additional setting from the Getting Started example will be the choice of high frequency or low frequency de-serialiser for the Camera Link interface.

High or Low Frequency De-serialisation

The example is set by default, to use low frequency de-serialisation. That is, the logic used to de-serialise the incoming camera data will only operate at pixel clocks up to 24MHz. Above this frequency the high frequency de-serialiser logic must be used.

To use the low frequency de-serialiser set `USE_LOW_FREQ_DESERIALISER` to `TRUE`. In this case, the Camera Link pixel clock frequency (Xclk) must be below 24MHz. To use the high frequency de-serialiser set `USE_LOW_FREQ_DESERIALISER` to `FALSE`. In this case the pixel clock frequency (Xclk) must be above 24MHz.

Generating a Bitstream

With the project created in the previous sections open in the ISE project navigator first highlight the top level design file '`top.vhd`' in the '`Module View`' window.

Double click on the 'Generate Programming File' process in the Process View window to begin creating a bitstream. This will trigger the following activity:

> Complete synthesis, using all of the project's source files. Since warnings are generated at this stage, you should see a yellow exclamation mark appear besides the "Synthesize" item in the Processes window.
>
> Complete Implementation:
> - Translation
> - Mapping
> - Placing
> - Routing
>
> Creation of the bit-stream:
> Note that this stage runs a Design-Rule-Check (DRC). The DRC can potentially detect anomalies created by the Place-and-Route phase.

When the processing ends, the proper bit-stream file, with extension "`.rbt`" can be found in the project directory.

To check that the User Constraints have been correctly applied to project check some of the entries in the PAD report file of the place and route stage against the location constraints in the project's UCF file. Every entry in this file should match the placement of the place and route tool. If you see different assignments, STOP HERE, and verify the UCF file selected for the project.

You can now download this file on your FPGA board and see how it works.

Checking the Module Vcco Setting

FPGA modules often include a jumper to define the Vcco for particular banks of Digital I/Os. If you are using such a module you will need to check that the correct Vcco selection has been made before running the Camera Link example on your FPGA module.

Running the Bitstream

Note that the user_ap design includes a very large counter, which divides the main system clock and drives LED 4. It is then obvious to see if the part has been properly programmed and downloaded: the LED should flash.

If the LED does not flash, try a hardware reset. The DLL used in the clock generation often does not start until a hardware reset. If the LED still does not flash we recommend that you shut down the PC or reprogram the device using a "safe" bit-stream. Otherwise, some electrical conflicts may happen (see below).

Possible causes for the device failure to operate are:

1. Wrong (or no) UCF file. This happens (for example) if you select the XST-version of the UCF with a Leonardo Spectrum (or Synplify) synthesis. The pin assignment for all the vectors (busses) will be ignored, and these pins will be distributed in a quasi-random fashion!
2. Wrong parameters in the CONFIG package.
3. Design Error.

If nothing seems obvious, re-run the camera example, or return to the Getting Started example.

The Example IP

Camera Inputs

The first major functional block in the Camera example is the VHDL module CLINK. This module is used to directly interface to the Camera Link interface of the camera that you are using. This module expects as inputs, four 7:1 serialised camera data signals, and a camera clock. From the four serialised data streams, the module recovers the original 28-bits of camera-link camera data.

From the 28-bits of data, 24-bits of camera image data are obtained, along with a data valid, line valid and frame valid control signals. The 24-bits of camera data are presented in the bottom 24-bits of the 32-bit data output by the CLINK module (data_out (31 downto 0)). The top byte is set to zero.

The control signals are used to generate a data-valid signal (dvalid_out), end of line control signal (eol_out) and end of frame control signal (eof_out), along with the pixel clock output (pixel_clock).

The CLINK module provides the capability to perform a byte shift so that the 8-bits of camera data processed by the FPGA match the top 8 active bits from the Camera Link camera. For an 8-bit camera, this byte shift is not required. However, when using 10-bit or 12-bit cameras, a byte shift right of 2 or 4 bits respectively is required to ensure that only the top 8-bits of each pixel data are processed.

Automatic Region of Interest Detection

Data output by the CLINK component is monitored by an automatic region of interest function. This process counts the number of pixels in a line and number of lines in a frame to create the Pixel-Total and Line-Total values.

The process also detects whether there are any dark (black) pixels on the left side, right side, top or bottom of the image, creating First-Pixel, Last-Pixel, First-Line and Last-Line values respectively.

These six values can be read using the HSB message interface. This allows controlling software to automatically detect where the active area of the image is. The values returned by the Automatic Region-of-Interest can be used in calculating the values with which to program the Capture Region of Interest.

The advantage of doing so over calculating the values by hand is that correct frame capture can be guaranteed. For example, if a camera is operating with 400 pixels in a line and 500 lines in a frame, and the region of interest and HERON FIFO transfer process are expecting 512 pixels x 512 lines from hand calculated values, then the capture process will never complete. This is because more data is expected than can be generated in one frame by the camera.

Using the automatically calculated values, the frame capture should always complete, as the software will be working with a frame size that is achievable. This is because the frame size used has been calculated from monitoring what the camera produces.

Capture Region of Interest

Data output by the CLINK component is put through a programmable region of interest. The module ROI performs a region of interest operation by using a pixel counter and a line counter, along with a pixel start position, pixel stop position, line start position and line stop position.

Starting with the first pixel in each line of data, the pixel counter starts at 0 and counts up. If the current pixel count is equal to or greater than the pixel start value, AND if the pixel count is less than or equal to the pixel stop value then that pixel is output. All other pixels in the line will be discarded. Similarly, starting with the first line in each frame of data, the line counter starts at 0 and counts up. If the current line count is equal to or greater than the line start value, AND if the line count is less than or equal to the line stop value then that line is output. All other lines in the frame will be discarded.

In the example the pixel start, pixel stop, line start and line stop values are all programmed over the HSB message interface. The values used can be calculated from the values returned by the Automatic Region of Interest logic.

Frame Control

The component FRAME_CONTROL controls when frames are output over the HERON FIFO Write Interface. The FRAME_CONTROL component can operate in two ways. It can be set-up to continuously output frames by setting the 'continuous' input high, or can be set-up to output between N frames (where N is 1 to 15) by setting a value on the 'frames' bus input and asserting the load signal.

After the region of interest has processed each frame, whether that frame will be output or not depends on the frame-control component. In the example the continuous mode and N-frames mode are programmed over the HSB message interface.

Output Packing and Synchronisation

The component PACK_SYNC takes the frames that are output by the FRAME_CONTROL component and packs the data ready for transmission through the HERON FIFOs. How the data is packed will depend on the mode of camera operation.

The pixel stream that is processed by the CLINK, ROI and FRAME_CONTROL components is a 32-bit wide data stream. This data stream may contain valid data only in the bottom byte, or only in the bottom two bytes or in all four bytes.

For 4x8 mode, only the bottom data byte carries camera data. In this mode, each 32-bit word sent to the HERON FIFOs is packed with four bytes of data. The data in the bottom byte is the first pixel in time, and the data in the top byte is the last pixel in time. For 2x16 mode each 32-bit FIFO word is packed to contain two 16-bit pixels. The bottom half of the word will contain the first pixel in time. Finally, in 1x32, one pixel is output in one 32-bit FIFO word. The 4x8, 2x16 and 1x32 mode control inputs are configured over the HSB message interface.

As well as data packing according to mode, this component also adds synchronisation markers to the data. One word, equal to 0 is added at the end of each camera line, apart from the last line of a frame, where one word with all bits set high (the value FFFFFFFFh) is added to indicate end of frame. In order to avoid the synchronisation values being present in the data, the bottom byte of camera data is modified to prevent the values 00h and FFh (a value of 00h will become 01h and a value of FFh will become FEh).

CoreGen FIFO and HERON FIFO Interface

After the data has been processed by the output packing and synchronisation component, the data is fed into a Core-generator generated FIFO. The FIFO is a 32-bit word, 511 word deep asynchronous FIFO built using Block RAM.

When there is any data in the CoreGen FIFO, it is read out and placed in the HERON FIFO Write Interface. The FIFO number used for output is programmed over the HSB message interface. If the HERON FIFO becomes full, eventually the CoreGen FIFO will become full and camera data will be lost. If the CoreGen FIFO becomes full, LED0 will become illuminated.

RS-232 Interface

The component RS232_UART interfaces HSB to the RS-232 camera control signals included in the Camera Link connection. This connection includes one signal for transmitting information to the camera, and one connection for the camera to transmit back.

On the FPGA to Camera side, there is a 16-byte transmit buffer that can be filled via HSB. As soon as data is placed in this buffer, it is automatically output to the camera at a Baud rate set in the Baud Rate register. This Baud rate can be selected in five incremental steps from 4800 Baud to 115200 Baud.

The transmit buffer has a TX count that indicates how many free spaces are available for data. By reading the TX count value over HSB, controlling software can calculate how many bytes can be loaded into the buffer in one go. Following a system reset, the transmit buffer will be empty, and the TX count will return 16. Data must not be sent to the transmit buffer while TX count is 0.

On the Camera to FPGA side, there is a 16-byte receive buffer that can be emptied via HSB. The receive buffer has a RX count that indicates how many bytes are available in the receive buffer. By reading the RX count value over HSB, controlling software can calculate how many bytes can be read from the buffer in one go. Following a system reset the receive buffer will be empty, and the RX count will return 0. Data cannot be read from the receive buffer while RX count is 0.

The particular sequence of values that must be sent or received using the RS-232 interface will vary from camera to camera. Please ensure that when using this interface, the values sent match those used by your camera type.

Typical FPGA Resource Usage

It may be interesting for someone who wants to develop their own FPGA design based on the example to judge how much of the FPGA is used by the camera example logic. There are also Block RAMs used by the CoreGen FIFO, but a newly developed design could replace or re-use that FIFO.

Xilinx FPGA	Typical Resource Usage
Virtex-II 1M gate FPGA	6%
Virtex-II 3M gate FPGA	2%
xc2vp7 Virtex-II Pro FPGA	20%
xc4vfx12 Virtex-4 FPGA	20%

Note: these figures are taken from a design that is not close to utilising the full FPGA. As you approach full utilisation the tools will probably pack the IP into a smaller percentage of the FPGA.

Developing the Camera Link Example

The Camera Link area-scan example is organised as a sequence of inter-connecting function blocks that each perform a separate process on a stream of camera data.

The functional blocks in the example provided include a block to interface directly to the camera signals, an automatic region of interest detection block, a block to reduce the image through a 'Region-of-Interest', a block to control when frames of data are output, and a block to pack and format the data ready for the next stage of processing.

Each of these blocks could be easily replaced, or removed to perform a different set of processing on the camera data.

For example, you may wish to perform no region of interest, and simply pass the entire camera frame from the camera directly to the next stage of processing. In this case, you could delete the `ROI` component instantiation, and connect the output of the `CLINK` component to the input of the `FRAME_CONTROL` component. This would be possible because the pixel-data-and-control interface of these three components has been designed to be the same.

When developing a new FPGA program for the HERON-FPGA module, by making a new component that interfaces to camera data in the same way, you will be able to easily insert that component into the processing chain that currently exists in the area-scan example.

Using the Pipelined-Pixel-Stream Format

The components `ROI` and `FRAME_CONTROL` of the area-scan example process data in a format that we will call the pipelined-pixel-stream format. As well as these two components, the `CLINK` component is used to take real-world camera signals and create an output stream that is in the pipelined-pixel-stream format, and the `PACK_SYNC` component takes data in that format and outputs as 32-bit words to a HERON FIFO.

This format is not a proprietary format, and can be changed in your design if necessary. It has simply been created so that a consistent starting point exists for fitting together separate functional units in an imaging application.

At any point in a pipeline that is using the 'pipelined-pixel-stream' format, there must be four control signals and N-bits of data. The control signals are `PIXEL_STROBE`, `DVALID`, `END_OF_LINE` and `END_OF_FRAME`. For the example provided, N is set to 32.

The `PIXEL_STROBE` is the pixel strobe or clock signal that must be used to clock all synchronous elements in the pixel processing pipeline. This signal will be typically derived from a clock signal output by the camera.

The `DVALID` signal is a 'data valid' signal. When asserted (set high) it must indicate that on a rising edge of the `PIXEL_STROBE`, there is valid data on the 32-bit data bus.

The `END_OF_LINE` and `END_OF_FRAME` signals are used to 'frame' the camera data. After the transmission of the last pixel in a line, the `END_OF_LINE` signal must be asserted (set high) for one pixel-clock period. Similarly, after the transmission of the last pixel in the last line, the `END_OF_FRAME` signal must be asserted (set high) for one pixel-clock period.

The signals `DVALID`, `END_OF_LINE` and `END_OF_FRAME` must be asserted in such a way that only one of them is high on any clock edge.

Changing the CLINK Component

The `CLINK` component performs an important task in the Camera Link example, in that it takes the signals that are provided by the Camera Link interface, and fits them to the format of the pipelined-pixel-stream used by all of the other components. If you are working with a camera format that is not supported by the CLINK component provided, you will need to change or replace this component.

The component will typically need to register the signals on the connectors in IOB registers inside the FPGA to meet the timing requirements of the camera output interface. It will then need to generate the PIXEL_STROBE, DVALID, END_OF_LINE and END_OF_FRAME signals as well as N data bits, where N will depend on the width of the camera data. The component must ensure that the timing of the DVALID signal directly matches the timing of the data that is output.

Changing the AUTO_ROI Component

The Automatic Region-of-Interest component provided implements logic that monitors how many pixels are output by the camera in each line, and how many lines are output in each frame. The component also automatically detects the left hand edge, right hand edge, top edge and bottom edge of the image so that dark pixels can be removed from the image by the capture region of interest component.

As this component only monitors the activity of the data pipeline, and does not directly modify signals that feed into downstream components, it can be modified or removed from the design without effecting the image processing.

Changing the ROI Component

The Region-of-Interest component provided implements a region of interest by using a start and stop position in a line, and start and stop position for lines in a frame. Essentially, this component will reduce, in each frame, the number of clock cycles where the DVALID signal is asserted. For incoming pixels within the region-of-interest, the DVALID output will be high when the DVALID input is also high. However, for a pixel that falls outside the region of interest, the DVALID output will always be low.

The END_OF_LINE and END_OF_FRAME signals must be preserved by the region of interest so that any processing units downstream will still be able to reconstruct the image.

Changing the FRAME_CONTROL Component

The frame-control component uses the END_OF_FRAME signal to decide when a new frame is being received. It combines the information provided by this signal with logic that defines which frames to output. While a frame is to be output the signals DVALID, END_OF_LINE and END_OF_FRAME, must be passed through un-altered. For any frame not to be transferred all control signals must be de-asserted for the whole of that frame.

Changing the PACK_SYNC Component

The output packing and synchronisation component receives data in the pipelined-pixel-stream format, by outputs data in a format that is defined inside the component. In the area-scan example provided, this format is made to suit transmission of an image over a HERON Output FIFO. Data is packed into words, and synchronisation markers are added to enable receiving software to reconstruct the image.

This component can be made to do whatever you want, as the contents of the component will be decided by the format of the data that you wish to output.

Adding New Components

New components can be added to do you want, and can be made to interface to the camera data in any way you choose. However, if you want to add new components and want to use the pipelined-pixel-stream format, then you must consider the following points.

- The `PIXEL_STROBE` signal must be used to clock all synchronous elements that interface to the camera data and control signals `DVALID`, `END_OF_LINE` and `END_OF_FRAME`.
- All valid pixels input to the component will be qualified by the input `DVALID` signal being asserted (set high). An output `DVALID` signal must be generated for the component. For each valid pixel output by the component, the output `DVALID` signal must be asserted.
- The input signals `END_OF_LINE` and `END_OF_FRAME` indicate where a line ends and where a frame ends respectively. These signals must be passed through unaltered, apart from the case where whole frames of data are to be removed from the data stream. In this case, the signals may be de-asserted throughout the frames that are removed.
- The data output by a component must be valid in the same clock cycle as the assertion of the `DVALID` signal to which that data corresponds.

What to check if you can’t get the image you expected

It may be possible that when you ran the example bitstream you were unable to capture an image, or that an image was captured but it was entirely dark, or that the image was smaller than expected. In either case, please read through the rest of this section, which is intended to help you find the cause of the problem.

Problem: The FPGA module is not outputting data

If the FPGA is not outputting any frames of data work through the check list below which describes reasons why this may have happen:

1. The camera you are using is not powered up at all, or is not powered up correctly. Please check the power supply that the camera is using and if necessary refer to the User Manual for your camera. In some cases, the camera has an LED on the back of the camera that when illuminated indicates the camera is correctly powered.
2. The camera is not correctly connected to the FPGA module. Please check that you have correctly connected the signals of the camera to the appropriate signals on Connector A and Connector B of the HERON-FPGA module. The Connector signal assignment information at the beginning of this document should be used when checking the camera signal connections.
3. The camera is operating with a set of control signals different to what is expected by the HERON-FPGA module. Please check the User Manual for the camera type you are using. The example IP is configured for a typical Camera Link camera. As such, it expects that the Camera Link Camera is providing a FVAL frame-valid signal that is active high (set to 1 when outputting valid frames) and a LVAL line-valid signal that is active high (set to 1 when outputting valid lines).
4. The FPGA requires that LVDS input signals are terminated with 100-Ohm resistors between the positive and negative signals of each pair. If you told HUNT ENGINEERING that you would use your module with a Camera Link camera, the correct resistor packs should have been fitted in the factory. If your HERON-FPGA module has not been fitted with the required 100-Ohm termination resistor packs as described in the section ‘Signal Voltage Levels’ earlier in this document the image will not be correctly captured. Please contact HUNT ENGINEERING to discuss having these components fitted to your module.

Problem: An Image was captured but was entirely dark or smaller than expected

If the example IP created images that were entirely dark or smaller than expected then this typically indicates some fairly simple errors that can quickly be fixed:

1. The image appears entirely black. Please check that the Lens cap has been removed.
2. The image appears very dark. Please open the Lens iris to let in more light.
3. The image is smaller than expected. This can be fixed in two ways. Firstly the example program is configured to use the automatic region of interest logic inside the FPGA. The automatic region of interest logic requires the camera to be generating images that are not very dark. In the case where the camera is generating dark images, the automatic region of interest may report a smaller frame size than is actually possible with a bright image. The solution is to open the lens iris to increase brightness of the image.