NEW ELECTRONIC TECHNOLOGY

Open Camera Platform Description

GigE PRO

GigE Vision cameras featuring integrated custom image pre-processing

Rev. 1.1-1702

Table of Contents

GigEPRO Open Camera Concept

Overview

NET´s GigEPRO cameras feature the Open Camera Concept which enables customers to implement own algorithms into the camera for unique product solutions.

GigEPRO is offered with a programmable FPGA (versions with Xilinx Spartan6 LX75/LX100) to allow image processing functions in real-time with a low deterministic processing delay. The FPGA is best capable of processing point and neighborhood operators of a typical image pre-processing task (and even simple interpretative tasks like feature extraction).

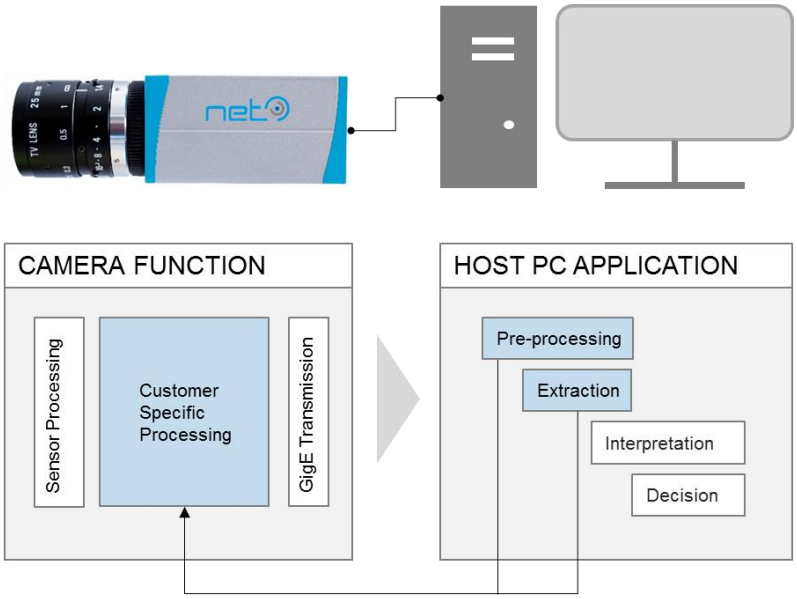

Figure 1: Partitioning of image processing tasks between camera and host

In order to realize an unique camera with powerful real-time image pre-processing NET provides customers with the documentation, camera hardware and software necessary to begin instantly with the customization of cameras.

Typical functions implemented by customers can be amongst others:

- image pre-processing
- image encryption / compression
- extraction, object detection / feature analysis
- 2D algorithms, point-to-point operations, e.g. image substraction
- laser scan line analysis
- optical trigger

Scope of Delivery

The scope of delivery for the customization of GigEPRO is as follows:

- GigEPRO camera with LX75 or LX100 FPGA incl. JTAG cable
- JTAG platform cable interface
- GigEPRO operational manual
- SynView, camera SDK with tools and documentation
- Open Camera Development Kit (ODK) including example designs and documentation

The following hardware and software items are required for the use of the Open Camera Development Kit, but are **not** provided by NET:

- X86 compatible PC with Linux, Windows XP or Windows 7 (required by XILINX ISE) as operating system
- XILINX Platform Cable USB II (in case of recovery of GigEPRO if GigE Vision stack is damaged as well as for FPGA and FLASH configuration, debugging and console)
- XILINX ISE 14.7 Design Suite with free WebPack license (LX75) or full license (LX100)
- GigE PoE adapter or a separate power supply of 9 to 24 VDC
- network cable CAT5e, CAT6 or higher

Figure 2: Open Camera Development Kit

System Overview

NET´s GigEPRO cameras and the SynView SDK package are fully compliant to GigE Vision, GenICam and GenTL standards. The camera platform is based on Xilinx Spartan6 LX45/LX75/LX100 FPGA devices. The basic camera design almost fills the LX45 FPGA. The additional capacity of the LX75 and LX100 FPGA devices, as well as the remaining DDR memory space allows users and system integrators to add a Custom Module with proprietary image processing functionality to the camera and develop customized products with integrated non-standard features.

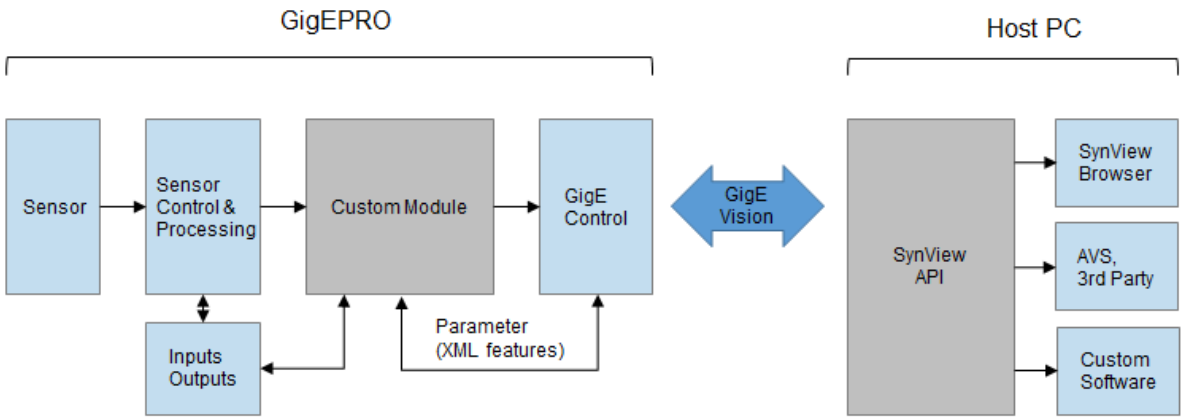

Figure 3: Open Camera Application

NET provides the tool chain setup for the FPGA code generation based on Xilinx ISE with the integrated Custom Module. The source code for several Custom Module example functions is available. The final FPGA design can be downloaded to the flash memory of the GigEPRO camera.

Camera Hardware Platform

The Open Camera platform allows integration of a customized function into the standard GigEPRO design. The hierarchical position of the Custom Module inside the camera design is fixed and cannot be changed.

The sensor image data pass through the basic GigEPRO pixel processing block, which contains gain/offset compensation, defect pixel correction, flatfield correction, debayering and color correction in case of color cameras, gamma correction or programmable LUT function, color space conversion and geometric correction. The resulting data is provided at the image input port of the Custom Module at 125 MHz system clock. The Custom Module image output port is connected to the frame buffer controller, which writes data to the DDR memory. Processed images are then automatically transmitted using the GigE transmitter.

The functionality of the Custom Module can be controlled with a set of integrated registers, which are connected to the embedded CPU via the Control Bus. The host application software can access these registers via the GenICam control channel protocol and device features, defined in the cameras XML file. 16 configuration registers (32 bit wide) for the Custom Module are automatically stored in the non-volatile user settings.

The Custom Module has direct access to one of the DDR memory manager read/write ports. Image and processing data can be transferred to the external DDR3 memory with respect to the overall memory bandwidth. For monochrome cameras full frame rate memory read and write of each image is supported. The Custom Module has direct access to the three I/O lines of GigEPRO. The customized FPGA design can be downloaded via JTAG for debugging purpose. The final FPGA design can be downloaded in the flash memory of the GigEPRO.

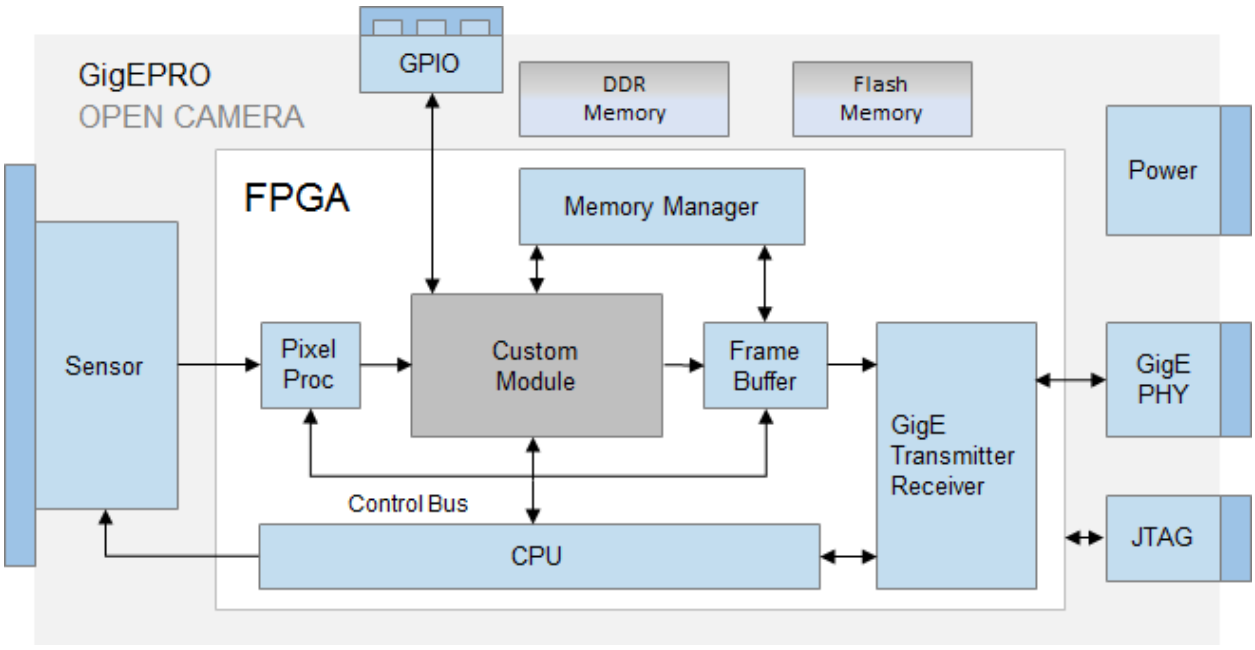

Figure 4: GigEPRO hardware platform

FPGA Resources

The standard GigEPRO camera design for the GigE Vision functionality occupies only a fraction of the FPGA. NET offers three size and performance options of the GigEPRO internal FPGA (LX45, LX75 and LX100). The LX75 and LX100 version have enough spare capacity to allow implementation of customized image processing tasks.

The tables below lists the available resources in the two FPGA types available for customer image processing.

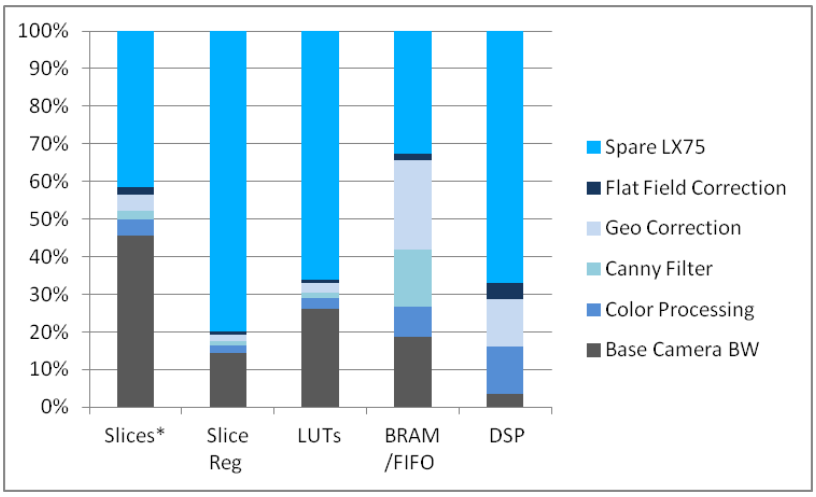

Figure 5: LX75 resources

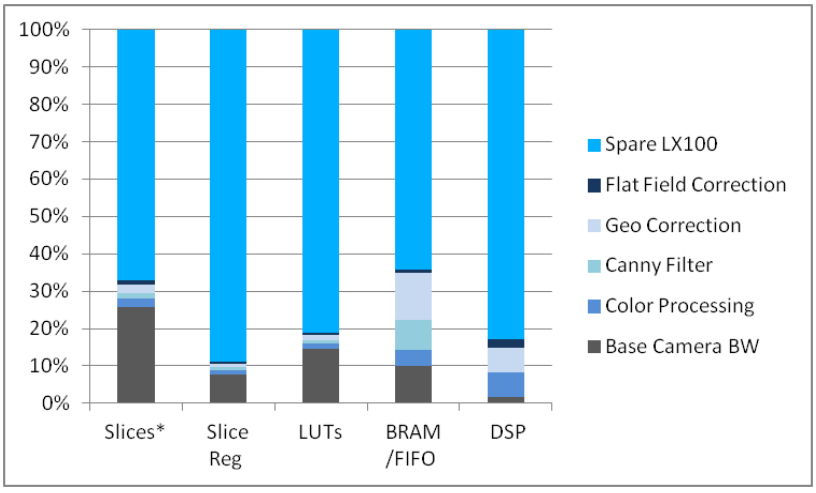

Figure 6: LX100 resources

Custom Module Design

Custom Module Interfaces

The FPGA netlist of the standard GigEPRO camera provides defined interfaces for the integration of the Custom Module HDL design. Sensor image data is available at the Image Receive Port. Processed image data is written to the Image Transmit Port. Internal control registers of the Custom Module are connected to the Control Bus Port. The internal CPU (32 bit MicroBlaze) is bus master on the Control Bus Port and transparently maps GenICam XML feature access via the GigE control channel to the Custom Module registers. The GPIO Port provides access to the camera GPIO lines. One 32 bit wide bi-directional DDR Memory Port is connected to the memory manager, allowing access to the external DDR3 memory. The System Port delivers clock and reset signals.

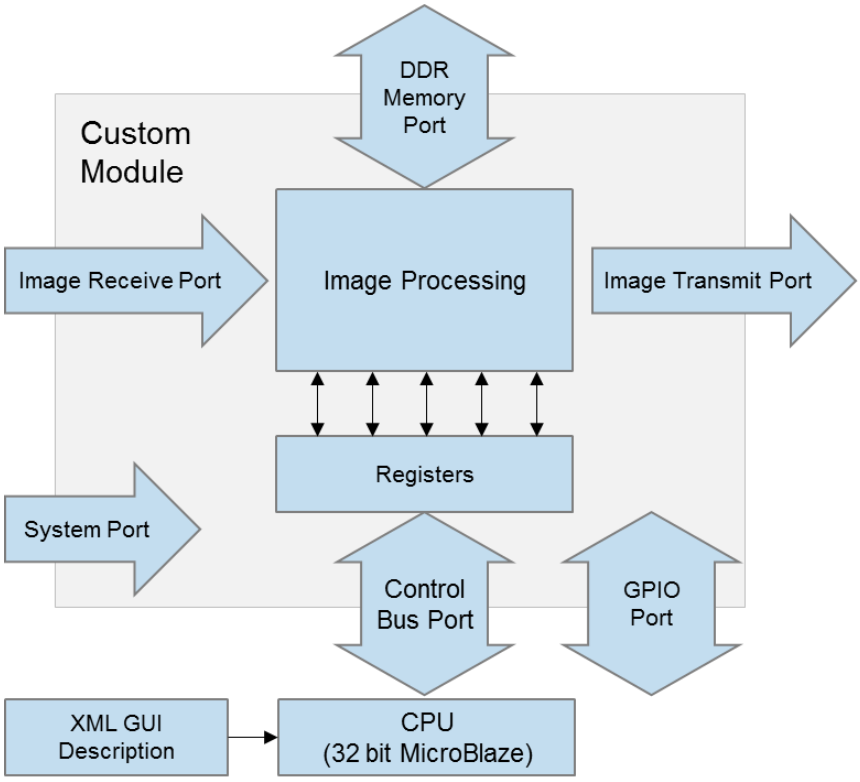

Figure 7: Custom Module interfaces

System Port

Verilog port definition:

```
//------------------------------------------------------------
// Module Infrastructure Signals
//------------------------------------------------------------
input       ImgRstxRI;         // Image Pipeline Reset
input       SysClkxCI;         // Custom Module Clock
input       SysRstxRI;         // Custom Module Reset
```

The **SysClkxCI** signal provides the 125 MHz clock signal for the image ports, the control bus and the processing inside the Custom Module. **SysRstxRI** is an asynchronous reset of the camera system during power up and reboot. The four cock cycle wide **ImgRstxRI** signal indicates the start of a new image acquisition cycle.

GPIO Port

Verilog port definition:

```
//------------------------------------------------------------
// GPIO Port
//------------------------------------------------------------
output [1:0]   CustIoCtrlxMO;     // GPIO Output Control
                                  // 00: CustIoDataOutxDO(1) disable
                                  //     CustIoDataOutxDO(0) disable
                                  // 01: CustIoDataOutxDO(1) disable
                                  //     CustIoDataOutxDO(0) enable
                                  // 10: CustIoDataOutxDO(1) enable
                                  //     CustIoDataOutxDO(0) disable
                                  // 11: CustIoDataOutxDO(1) enable
                                  //     CustIoDataOutxDO(0) enable
output [1:0]   CustIoDataOutxDO; // GPIO Output Data
input          CustIoDataInxDI;  // GPIO Input Data
```

The GPIO Port signals provide access to the GigEPRO external IO lines. With **CustIoCtrlxMO** the Custom Module can take over the control of the two output lines.

Image Receive Port

Verilog port definition:

```
//------------------------------------------------------------
// Image Receive Port
//------------------------------------------------------------
input  [35:0]   ImgRxPixDataxDI;      // 3x12 bit RGB pixel data
output          ImgRxPixRdyxMO;       // unused in GigEPRO
input  [2:0]    ImgRxPixStatxSI;      // image frame sync code
                                      // 001 Start-of-Frame (SOF)
                                      // 010 Start-of-Single-Line (SOSL)
                                      // 011 Start-of-Data (SOD)
                                      // 100 Start-of-Line (SOL)
                                      // 101 Start-of-Last-Line (SOLL)
                                      // 110 End-of-Line/Data (EOL)
                                      // 111 Data-Error (ERR)
input           ImgRxPixValxMI;       // data valid
```

The corresponding clock for the port signals is the 125 MHz clock from System Port. Frame sync codes and pixel data coding are described in the Image Port Description chapter. Pixel and sync date are valid only when **ImgRxPixValxMI** is high. The GigEPRO image ports do not support any data handshake. The **ImgRxPixRdyxMO** output signal should be set 1.

Image Transmit Port

Verilog port definition:

```
//------------------------------------------------------------
// Image Transmit Port
//------------------------------------------------------------
output [35:0]    ImgTxPixDataxDO;      // 3x12 bit RGB pixel data
input            ImgTxPixRdyxMI;       // unused in GigEPRO
output [2:0]     ImgTxPixStatxSO;      // image frame sync code
                                       // 001 Start-of-Frame (SOF)
                                       // 010 Reserved for future use
                                       // 011 Start-of-Data (SOD)
                                       // 100 Start-of-Line (SOL)
                                       // 101 Start-of-Last-Line (SOLL)
                                       // 110 End-of-Line/Data (EOL)
                                       // 111 Data-Error (ERR)
output           ImgTxPixValxMO;       // data valid
```

The corresponding clock for the port signals is the 125 MHz chock from System Port. Frame sync codes and pixel data coding are described in the Image Port Description chapter. **ImgTxPixValxMO** must be high for any valid pixel and frame sync code. The GigEPRO image ports do not support any data handshake. The incoming **ImgTxPixRdyxMI** signal should be ignored.

Image Port Description

Status information about the relative position of the pixel within a frame is defined by the image frame sync code. ImgStatus is a 3 bit bus and the decoding is shown in the table below:

ImgStatus(2..0)	Status Decoding
0	None (inside frame)
1	Start of Frame (SOF)
2	Start of Single Line Frame (SOSF) (LineScan)
3	(not used)
4	Start of Line (SOL)
5	Start of Last Line (SOLL)
6	End of Line (EOL)
7	Error (ERR)

The Start-of-Frame (SOF) status indicates the first pixel of an area scan image. If line scan mode is active, SOF is replaced with the Start-of-Single-Line-Frame (SOSF) status. The first pixel of every other line is labelled as Start-of-Line until the last line is reached. The first pixel of the last line is referred to as Start-of-Last-Line (SOLL). The last pixel of every line is referred to as End-of-Line (EOL). The smallest possible area scan image is a 2x2 image.

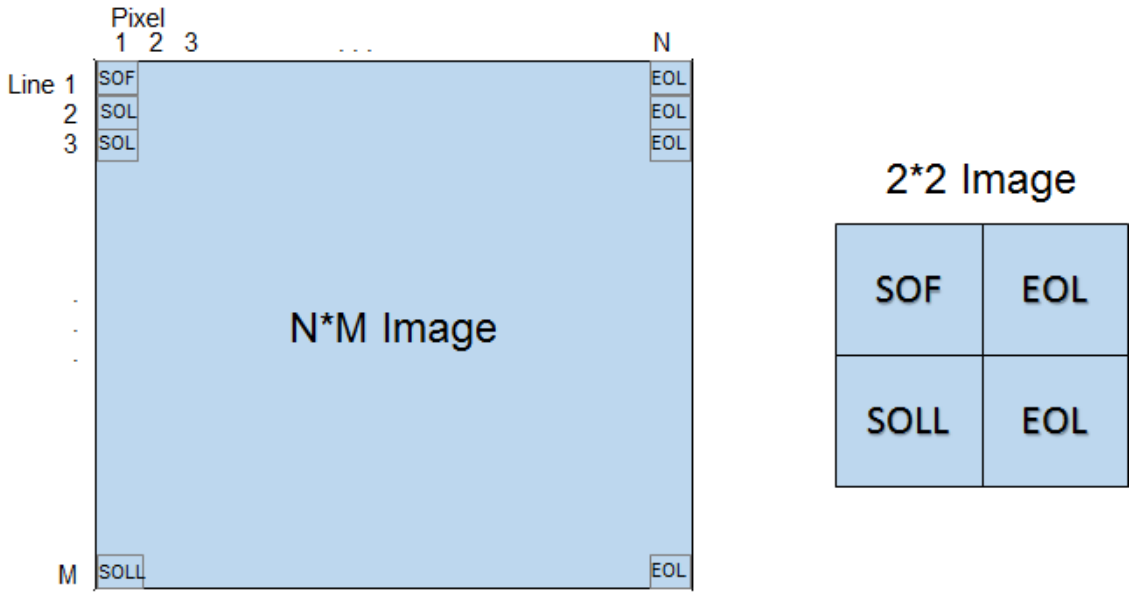

Figure 8: Image Frame Coding

The GigE transmission of the frame will start immediately after the end of the last line, indicated with SOLL and corresponding EOL. In this moment the GigE system time code, which is part of the frame header information, is latched.

The transmission of the Data-Error code (111) will prevent the GigE transmission of the started frame. The previously received part of the image will be dropped and the GigE transmitter waits for the next frame start code.

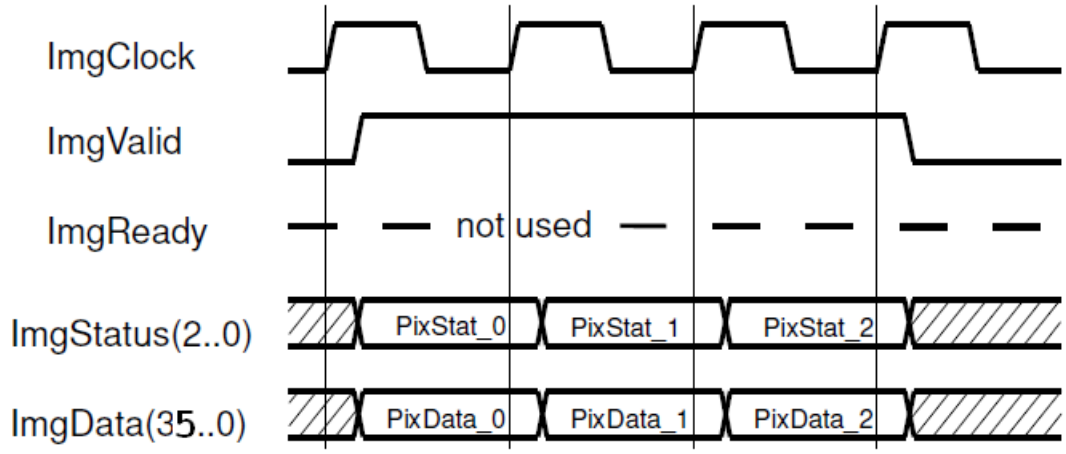

Figure 9: Image Rx/Tx Port

Raw or processed sensor data in RGB, Bayer or Mono format are mapped to the packed 36 bit data vector **ImgPixData[35:0]** according to the PixelFormat feature setting of the GigEPRO camera. 8/10-bit formats are MSB aligned.

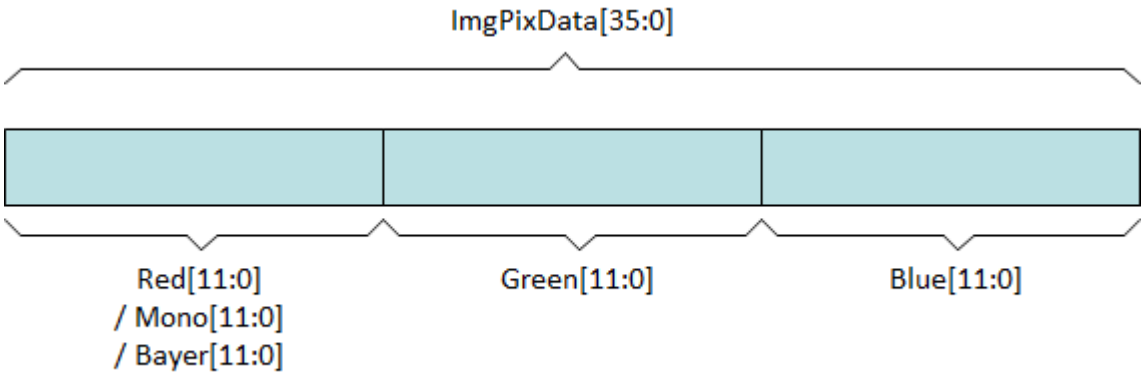

Figure 10: Image Bus Mapping

Control Bus Port

Verilog port definition:

```
//------------------------------------------------------
// Control Bus Port
//------------------------------------------------------
output          SysAckxMO;     // acknowledge for successful read/write
input  [16:0]   SysAddrxAI;    // read/write byte address
output          SysErrxMO;     // acknowledge for unsuccessful read/write
output [31:0]   SysRdDataxDO;  // read data
input           SysReqxMI;     // bus read/write request
input  [31:0]   SysWrDataxDI;  // write data
input           SysWrEnxMI;    // write enable
```

The GigEPRO embedded CPU communicates with the Custom Module via the control bus. The CPU acts as control bus master, whereas the Custom Module is control bus slave. The control bus performs read and write operations on 32 bit data register. The address space is 32k word, thus 32768 registers can be connected. The **SysAddrxAI** port vector contains byte address values, so the two least significant bits are always zero and thus they can be ignored, whereas the upper part represents the word address. Bus read/write operation is requested with the **SysReqxMI** signal. For read operations **SysWrEnxMI** is 0, for write operations **SysWrEnxMI** is 1. Any read and write requests must be acknowledged by the Custom Module with the **SysAckxMO** or in case of an error with the **SysErrxMO** signal. Access errors, indicated with **SysErrxMO** do not have any consequence in the GigEPRO system.

At word address 0 the Custom Module has to provide a 32 bit unique identification number in the range of 0x00000001…0xfffffffe. Registers that are connected to the control bus can be controlled via the standard GenICam XML configuration interface of the GigEPRO camera. Configuration data in the word address range 1…16 are part of the non-volatile camera user set, which is controlled with the GenICam UserSet features.

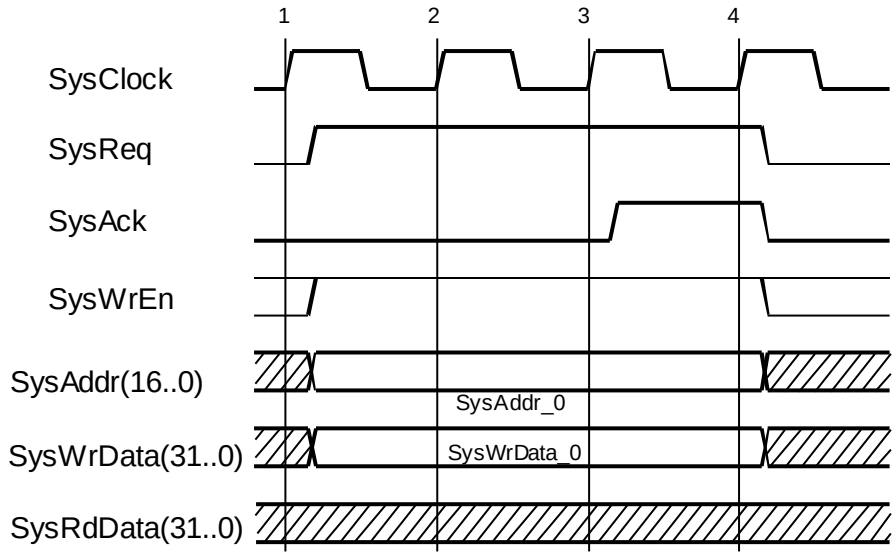

Figure 11: Control Bus Write Operation

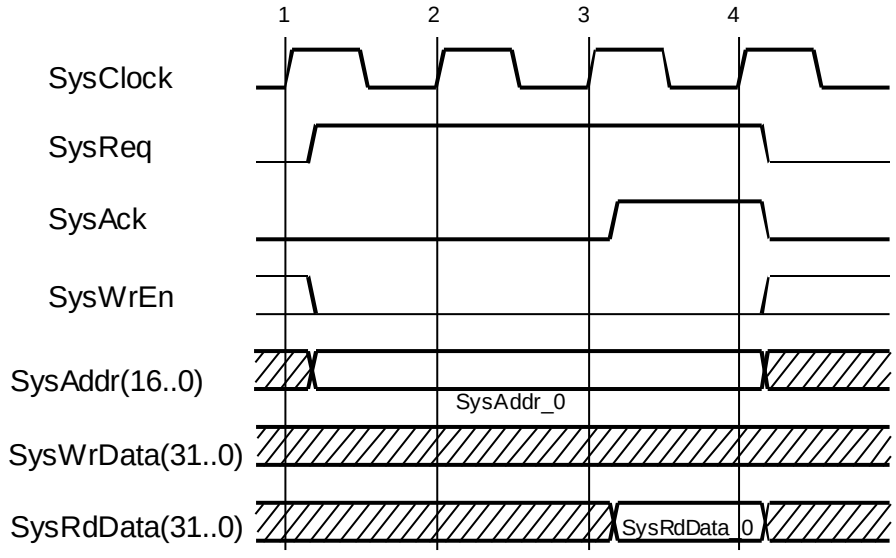

Figure 12: Control Bus Read Operation

DDR Memory Port

The DDR memory port directly connects one of the Xilinx Spartan6 MPMC memory manager MCB read/write ports (32 bit) to the Custom Module. The external DDR3 memory device is connected with 16 bit wide connection running at 312 MHz.

Verilog port definition:

```
//------------------------------------------------------------
// DDR Memory Port
//------------------------------------------------------------
// --- command ---
output [29:0]   px_cmd_addr_o;    // Byte start address for current
                                  // transaction. Addresses must be
                                  // aligned to port size:
                                  // 32-bit ports: Lower two bits must be 0
output [5:0]    px_cmd_bl_o;      // Burst length in number of user words
                                  // for the current transaction. Burst
                                  // length is encoded as 0 to 63,
                                  // representing 1 to 64 user words [for
                                  // example, b00011 is a burst length 4
                                  // transaction]. The user word width
                                  // equals the port width [for example,
                                  // a burst length of 3 on a 64-bit port
                                  // transfers 3 x 64-bit user words =
                                  // 192 bits total].
output          px_cmd_en_o;      // This active-High signal is the
                                  // write-enable signal for the Command
                                  // FIFO
input           px_cmd_full_i;    // This active-High output is the full
                                  // flag for the Command FIFO. It
                                  // indicates the FIFO cannot accept any
                                  // more commands and blocks writes to
                                  // the Command FIFO.
output [2:0]    px_cmd_instr_o;   // Command code for the current
                                  // instruction. Bit 0 represents the
                                  // READ/WRITE select, Bit 1 is Auto
                                  // Precharge enable, and Bit 2
                                  // represents Refresh, which always
                                  // takes priority:
                                  // Write: 3b000
                                  // Read: 3b001
// --- read ---
input  [31:0]   px_rd_data_i;     // Read Data value returning from
                                  // memory. This signal is driven by the
                                  // output of the Read Data FIFO into
                                  // FPGA logic. PX_SIZE can be 32, 64,
                                  // or 128 bits, depending on the port
                                  // configuration.
input           px_rd_empty_i;    // This active-High signal is the empty
                                  // flag for the Read Data FIFO. It
                                  // indicates there is no valid data in
                                  // the FIFO.
output          px_rd_en_o;       // This active-High signal is the read
```

```
                                    // enable for the Read Data FIFO. Read
                                    // Data is clocked out of the FIFO on
                                    // the rising edge of pX_rd_clk when
                                    // pX_rd_en = 1 and pX_rd_empty = 0.
// --- write ---
output [31:0]   px_wr_data_o;       // Write Data value to be loaded into
                                    // Write Data FIFO and sent to memory.
                                    // PX_SIZE can be 32, 64, or 128 bits,
                                   // depending on port configuration.
output          px_wr_en_o;        // This active-High signal is the write
                                   // enable for the Write Data FIFO. It
                                   // indicates that the value on
                                   // pX_wr_data is valid for loading into
                                   // the FIFO. Data is loaded on the
                                   // rising edge of pX_wr_clk when
                                   // pX_wr_en = 1 and pX_wr_full = 0.
input           px_wr_full_i;      // This active-High signal is the full
                                   // flag for the Write Data FIFO. When
                                   // this signal is high, it prevents
                                   // data from being loaded into the FIFO.
```

Custom Module Register

Word Address	CPU Access	Description	Update
Module ID			
0x0000	R	Custom Module ID Range 0x000001…0xfffffffe	boot
User Register (for communication between host application and Custom Module)			
0x0001 …0x0010	RW	Custom Module registers which can be stored in the non-volatile camera UserSet 1 or 2 and reloaded during reset.	
0x0011 …0x7f00	RW	Custom Module registers which can only be accessed by host application.	
Special Function Register (for communication between CPU and Custom Module)			
0x7f80	W	**MemStartAddr** Start address of the requested DDR memory. Valid only when **MemWriteEn** is set to 1.	boot
0x7f81	W	**MemWriteEn** Enable DDR memory access beginning at **MemStartAddr**	boot
0x7f82	W	**TimeStamp** 32 bit of the current GigE timestamp value	idle
0x7f83	W	**SensorWidth** GigE sensor width value of the camera	boot
0x7f84	W	**SensorHeight** GigE sensor height value of the camera	boot
0x7f85	W	**AquisitionRunning** Status of the GigE aquisition	start/stop
0x7f86	W	**ImageWidth** Current GigE image width value	start
0x7f87	W	**ImageHeight** Current GigE image height value	start
0x7f88	W	**PixelFormat** 32-bit current GigE pixel format value	start
0x7fc0	R	**MemSizeReq** DDR memory size request in number of bytes	boot
0x7fc1	R	**AutoSensorStart** When set camera will automatically start sensor image processing.	boot

FPGA Design Flow

Tool Chain

The Open Camera Development Kit (ODK) contains the required files for building the GigEPRO camera FPGA design with embedded Custom Module function. FPGA synthesis and programming file generation flow are based on Xilinx ISE tool chain. The Xilinx ISE 14.7 Design Suite has to be installed on the host PC. For Spartan6 LX75 devices the free Webpack license version can be used, for LX100 devices the full license version is required.

The design for the Custom Module has to be available as a set of HDL files (Verilog or VHDL), together with the corresponding ISE project file containing a list of the required HDL files for synthesis. Source code for several design examples are provided together with the Open Camera Development Kit.

After synthesis the Custom Module netlist is translated together with the provided GigEPRO toplevel netlist. The final `bitstream.bit` file can be used for JTAG FPGA configuration with the ISE Impact tool and the Xilinx Platform Cable USB II for debugging purpose. The `bitstream.bin` file can be used to download the final design into the GigEPRO flash memory.

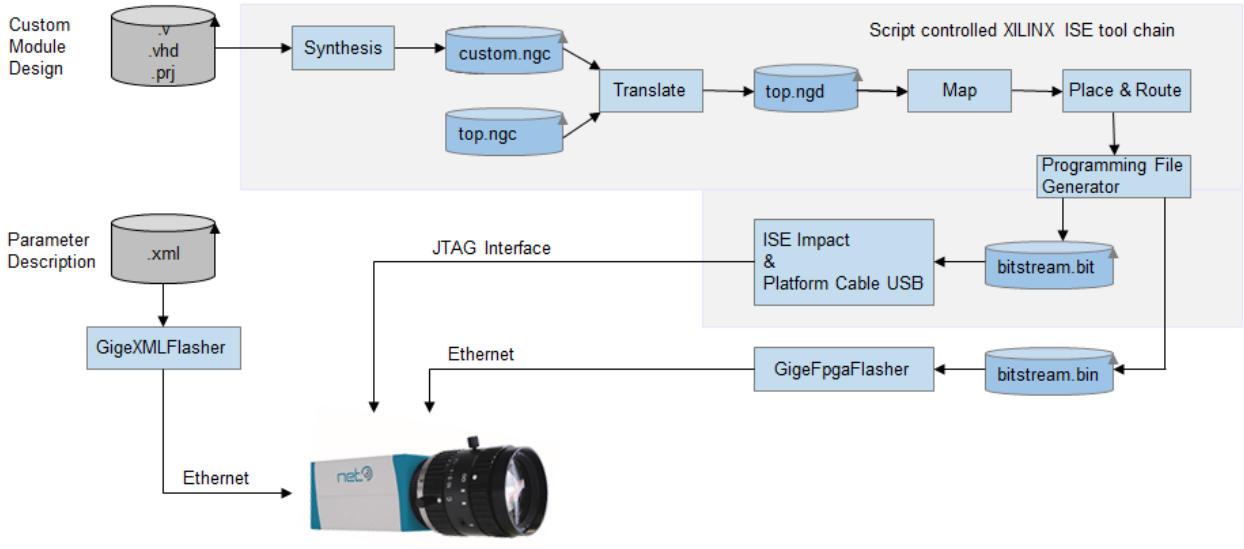

Figure 13: FPGA design flow

Directory Structure

The Open Camera Development Kit (ODK) is delivered with the directory structure depicted below. All files generated during synthesis flow stay inside this directory structure. Thus it is possible to have several parallel Open Camera projects at the same time.

```
GigEPRO-OpenCamera-2017-02-22     // ODK base directory
→ cores                          // core netlists
→ fw                             // bootloader
→ gige-dma-full-lx75_3
  → cpu                          // 32bit uBlaze netlist
  → src                          // ucf file
  → src_open_camera              // Custom Module project file
  → xst                          // GigEPRO top netlist
  → xst_open_camera              // "Canny" Custom Module netlist
→ src
  → open_camera                  // sources of Custom Module examples
  → Open_camera_tb               // simulation test bench
→ syn                            // synthesis scripts
→ tcl                            // scripts to simulate & debug
→ tools                          // tools for download & XML merge
  → data                         // original bitstreams for recovery
```

The delivered ODK already contains project files and netlists for the "Canny" Custom Module demo example to speed up first learning steps.

Synthesis Execution

The Xilinx ISE 14.7 Design Suite has to be installed on the host PC.

The HDL files for the Custom Module design have to be listed in the project file named `CustomModule.prj` placed in the `/ODK/gige-dma-full-lx???_?/src_open_camera` directory.

FPGA synthesis can be started from the Xilinx "ISE Design Suite Command Prompt". All commands have to be entered from the current working directory `/ODK/gige_dma_full_lx???_?`.

The command `../syn/syn.sh C0` clears all temporary files from previous sessions.
The command `../syn/syn.sh` starts the FPGA synthesis flow.

Warning: The Open Camera delivery from NET already contains predefined project files and netlists. Therefore a clean directory structure should be enforced by `../syn/syn.sh C0` before starting any work with a modified Custom Module.

After successful synthesis the directory `/ODK/gige_dma_full_lx???_?/bit` will contain the `bitstream.bit` and `bitstream.bin` files for FPGA download and configuration.

FPGA Configuration via JTAG

For FPGA development the GigEPRO camera is available with an additional connector for JTAG debugging. With the ISE Impact tool and the Xilinx Platform Cable USB II the generated `bitstream.bit` file can be used to download the FPGA configuration.

The camera will reboot right after configuration and the design can be tested. The FPGA configuration will be lost after camera reset or power up.

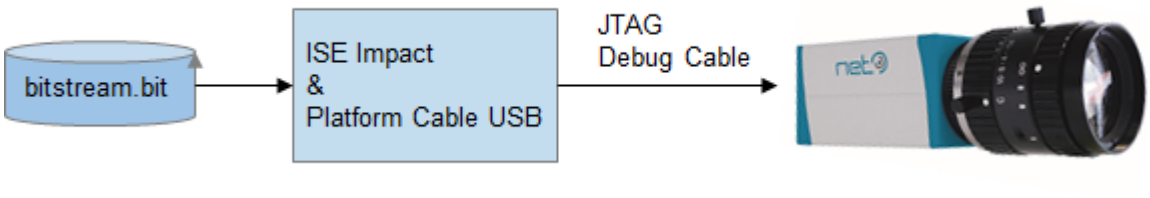

Figure 14: FPGA Configuration

FPGA Configuration Download to Flash

The generated `bitstream.bin` file can be downloaded into the GigEPRO flash memory for permanent camera configuration (the design should have been tested before by JTAG configuration). The Open Camera Development Kit contains the GigeFpgaFlasher tool for this procedure. The camera configuration is done via the Ethernet connection of the GigEPRO camera. For this the SynView SDK has to be installed on the host PC.

The win32 GigeFpgaFlasher.bin tool can be found in the /`ODK/tools/bin` directory. A batch file for direct execution can be found in the `/ODK/tools` directory.

The download tool has to identify the camera, therefore the actual IP address (or the model name) of the camera has to be listed in the batch file before execution.

LatestSynth_FPGAFlashingLx75.bat:

```
rem * Download FPGA Configuration to GigEPRO Flash Memory *

bin\GigeFpgaFlasher 192.168.0.222 ../gige-dma-full-lx75_3/bit/bitstream.bin bf
pause
```

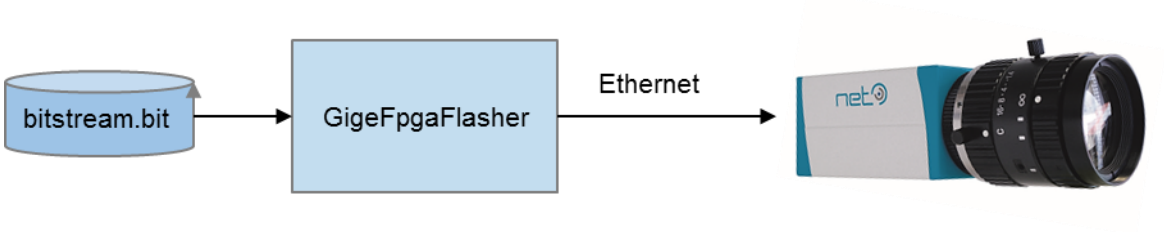

Figure 15: FPGA Flashing

Source Code Simulation

Testbench

The Open Camera Development Kit contains a testbench for the simulation of the Custom Module. The testbench consists of four modules (ImgBusStimulus, ImgBusAnalyzer, SysBusStimulus, mcb_stim) to stimulate and analyze the interfaces of the Custom Module and one module (SimParaInit) to configure the whole testbench.

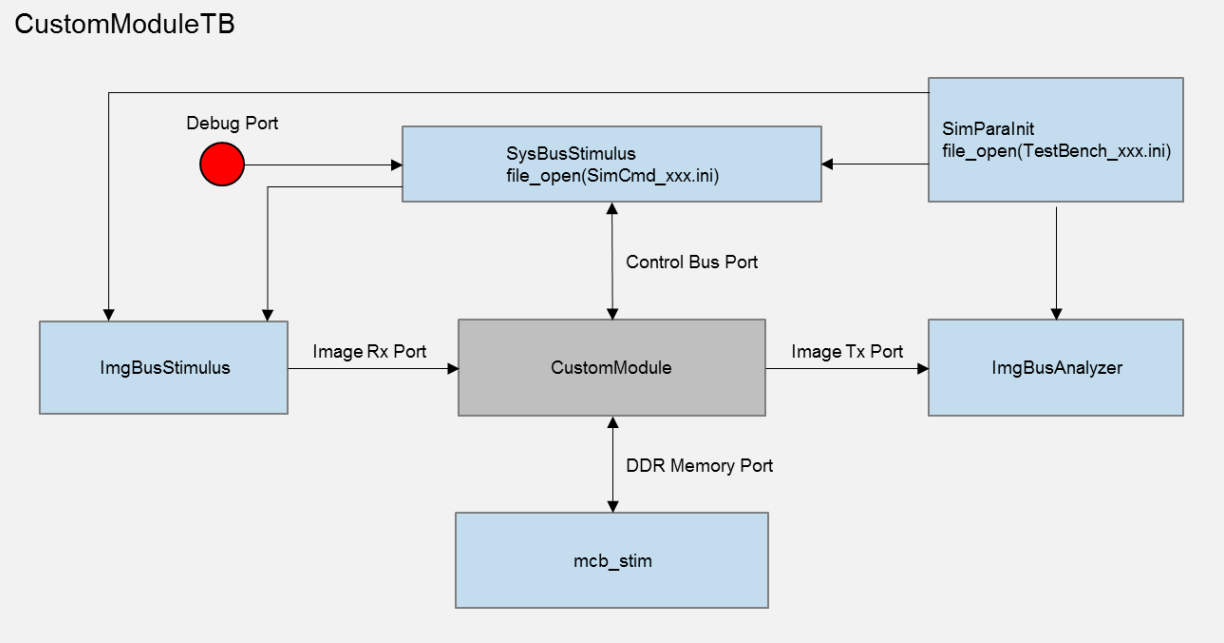

Figure 16: Custom Module Testbench

ImgBusStimulus reads the specified input image file and transmits the image data via *Image Receive Port* to the CustomModule.

ImgBusAnalyzer receives the image data via *Image Transmit Port,* compares the data with the reference image and saves the received image in the output image file.

SysBusStimulus will be controlled by SimCmd_xxx.ini. The module performs the communication with the Custom Modul via *Control Bus Port* and starts and stops the image transmission via ImgBusStimulus. Furthermore signals from the testbench can be connected with the *Debug Port* to use them in SimCmd_xxx.ini.

mcb_stim is a simple simulation model of the MCB-Controller. The default amount of allocated memory is 20000 DWORDs (DWORD = 4Byte) (see cMEM_SIZE in CustomCfgLib.vhd). The source files of the testbench can be found in `/ODK/src/open_camera_tb`. To control the testbench and to generate the simulation sequence two .ini-files are necessary:

- `TestBench_xxx.ini  (xxx = CustomModule)`
- `SimCmd_xxx.ini`

TestBench_xxx.ini specifies the general simulation and testbench parameter (e.g. the simulation time) as well as the file name of the input, output and reference image.

SimCmd_xxx.ini contains a sequence of the Control Bus commands and starts and stops the *Image Receive Port*.

The configuration files for the simulation of the “Canny” Custom Module example are located in `/ODK/src/open_camera/canny/tb_config`.

Execute Simulation

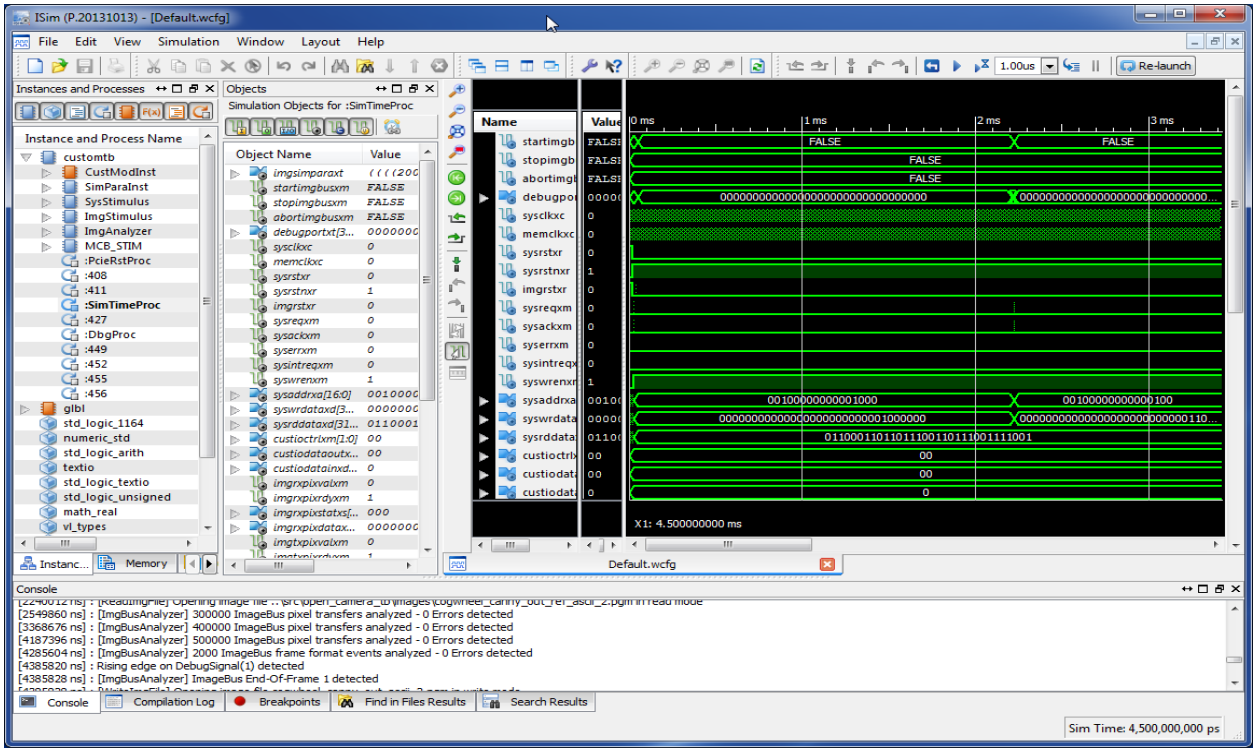

Figure 17: Custom Module simulation

To simulate the Custom Module with the included testbench the following steps are necessary:

- Open Xilinx ISE 14.7
- Set the current directory in the ISE Tcl Console to `/ODK/tcl`.
- Execute the Tcl-Skript “build_cm_ise.tcl” (-> source build_cm_ise.tcl)

An ISE project with the Custem Module testbench and the “Canny” design will be generated.

To start the simulation switch in the Design Tab to the Simulation View select “CustomTB” and run in the Xilinx ISim Simulator the “Simulation Behavioral Model”. The Xilinx ISim Simulator starts the execution of the testbench. RunAll starts the simulation until the MaxSimTime specified in `TestBench_xxx.ini` is reached.

If another Custom Module should be simulated, remove all source files from the “Canny” design and include the desired Custom Module to the ISE project. Additionally adapt the testbench configuration files to the needs of the new Custom Module:

- `CustomCfg_CANNY.vhd`
- `SimCmd_Canny.ini`
- `TestBench_Canny.ini`

Hint: Testbench configuration files of the other Custum Modules (scaled, diffpic) can be used as an example.

Hardware Debugging via ChipScope

To debug the Custom Module via Xilinx ChipScope the following steps are necessary to generate a camera bitstream with ChipScope included:

- Open Xilinx ISE 14.7
- Set the current directory in the ISE Tcl Console to `/ODK/tcl`.
- Execute the Tcl-Skript “build_gige_ise.tcl” (-> source build_gige_ise.tcl)

An ISE project with the all camera ngc-files (including the `CustomModule.ngc` located in `/ODK/gige-dma-full-lx75_3/xst_open_camera)` will be generated.

Now you can add (*Project -> New Source…*) a new “ChipScope Definition and Connection File” (.cdc-file). To configure the ChipScope core and select the debug signals you have to open the .cdc-file (Hint: Try it twice, for the first time it often fails).

The Custom Module signals are located in:

```
- [gige_top]
     - VIDEO_INST [video]
          -
      SENSPROC_TOP_INST/u_gige_sensproc/u_gige_rec/u_sif2cmif/u_CustomModule
      [CustomModule]
```

If ChipScope .cdc-file is included and configured execute “Generate Programming File” to start the implementation of the bitstream and load it with the Xilinx ISE Impact Tool into the camera.

To start the debugging and to make the debug signals visible open the Xilinx ChipScope Analyze Tool and connect the program via the Xilinx Platform Cable USB II to the FPGA.

Custom Module Design Examples

The Open Camera Development Kit (ODK) contains different design examples for the Custom Module. The source code for these examples can be found in the `/ODK/src/open_camera` folder.

Canny

This is the canny edge detector design, which is part of the standard GigEPRO camera design. HDL language is Verilog. Local RAM is used for line memories. Sensor data can be monochrome or RGB.

DiffPic

This design demonstrates the use of DDR memory. The data output is the pixel difference between actual and previous frame. Sensor data must be monochrome.

BlockDemo

This design generates RGB block pattern overlay. Customized XML description for control registers is provided. Sensor data can be monochrome or RGB.

Scale_D

This design allows 2-dimensional downscaling of the sensor data. This design demonstrates how to implement image processing with different input and output image size. Sensor data can be monochrome or RGB.

XML Feature Description

File Format

Being GenICam compliant, the GigEPRO camera contains a XML file with detailed description of the cameras features. The standard GigEPRO XML file already contains a category for accessing control registers of the Custom Module. As soon as the camera detects a valid Custom Module ID, the corresponding XML category will be visible. The standard XML file provides basic 32-bit Custom Module register access with address and data parameters.

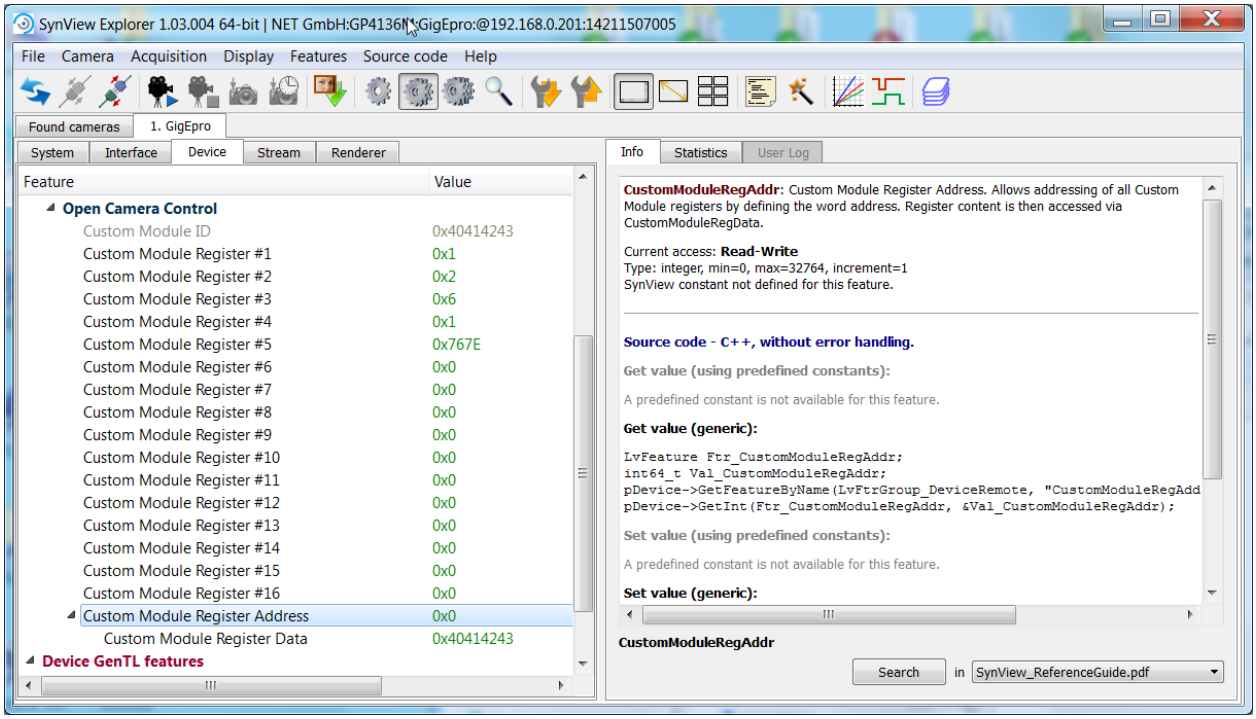

Figure 18: Custom Module Control

For a more user friendly representation of the embedded Custom Module design, the generic XML description for the Custom Module can be replaced by a customized XML feature description. The customized XML feature description has to be compliant to GenICam standards. The Custom Module registers are mapped to the global XML byte address space starting at a baseaddress which can be read via **CustomModuleRegBase** XML register. The Open Camera Development Kit contains a template XML description of the BlockDemo Custom Module design example (blockDemo.xml):

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<!-- +                 OSD Demo Register Description                    + -->
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<Category Name="OpenCamControl" NameSpace="Custom">
        <ToolTip>Category that contains the OSD control features.</ToolTip>
```

```
        <Description>Category that contains the OSD control features.</Description>
        <DisplayName>OSD Control</DisplayName>
        <Visibility>Beginner</Visibility>
        <ImposedAccessMode>RO</ImposedAccessMode>
        <pFeature>OsdCustomModuleName</pFeature>
        <pFeature>OsdCustomModuleId</pFeature>
        <pFeature>OsdEnable</pFeature>
        <pFeature>OsdSpeed</pFeature>
        <pFeature>OsdSize</pFeature>
        <pFeature>OsdColorEnable</pFeature>
        <pFeature>OsdFrameCount</pFeature>
</Category>

<Group Comment="OpenCamControl">

    <StringReg Name="OsdCustomModuleName" NameSpace="Custom">
        <Description>OSD Custom Module Name.</Description>
        <DisplayName>OSD Custom Module Name</DisplayName>
        <Visibility>Beginner</Visibility>
        <pAddress>CustomModuleRegBase</pAddress>
        <Length>4</Length>
        <AccessMode>RO</AccessMode>
        <pPort>Device</pPort>
    </StringReg>
    <Integer Name="OsdCustomModuleId" NameSpace="Custom">
        <Description>OSD Custom Module ID.</Description>
        <DisplayName>OSD Custom Module ID</DisplayName>
        <Visibility>Beginner</Visibility>
        <pValue>CustomModuleId_Reg</pValue>
        <Min>0</Min>
        <Max>0xffffffff</Max>
        <Representation>HexNumber</Representation>
    </Integer>
    <IntReg Name="CustomModuleId_Reg">
        <Visibility>Invisible</Visibility>
        <pAddress>CustomModuleRegBase</pAddress>
        <Length>4</Length>
        <AccessMode>RO</AccessMode>
        <pPort>Device</pPort>
        <Sign>Unsigned</Sign>
        <Endianess>BigEndian</Endianess>
    </IntReg>

    <Boolean Name="OsdEnable" NameSpace="Custom">
        <Description>OSD Enable.</Description>
        <DisplayName>OSD Enable</DisplayName>
        <Visibility>Beginner</Visibility>
        <Streamable>Yes</Streamable>
        <pValue>CustomModuleReg1_Val</pValue>
        <OnValue>1</OnValue>
        <OffValue>0</OffValue>
    </Boolean>
    <IntReg Name="CustomModuleReg1_Val">
        <Visibility>Invisible</Visibility>
        <IntSwissKnife Name="CustomModuleReg1Addr">
```

```
            <pVariable Name="CMBASE">CustomModuleRegBase</pVariable>
            <Formula>CMBASE+1*4</Formula>
        </IntSwissKnife>
        <Length>4</Length>
        <AccessMode>RW</AccessMode>
        <pPort>Device</pPort>
        <Sign>Unsigned</Sign>
        <Endianess>BigEndian</Endianess>
    </IntReg>
    …
```

This XML feature description file will generate the following OSD control in the camera XML tree:

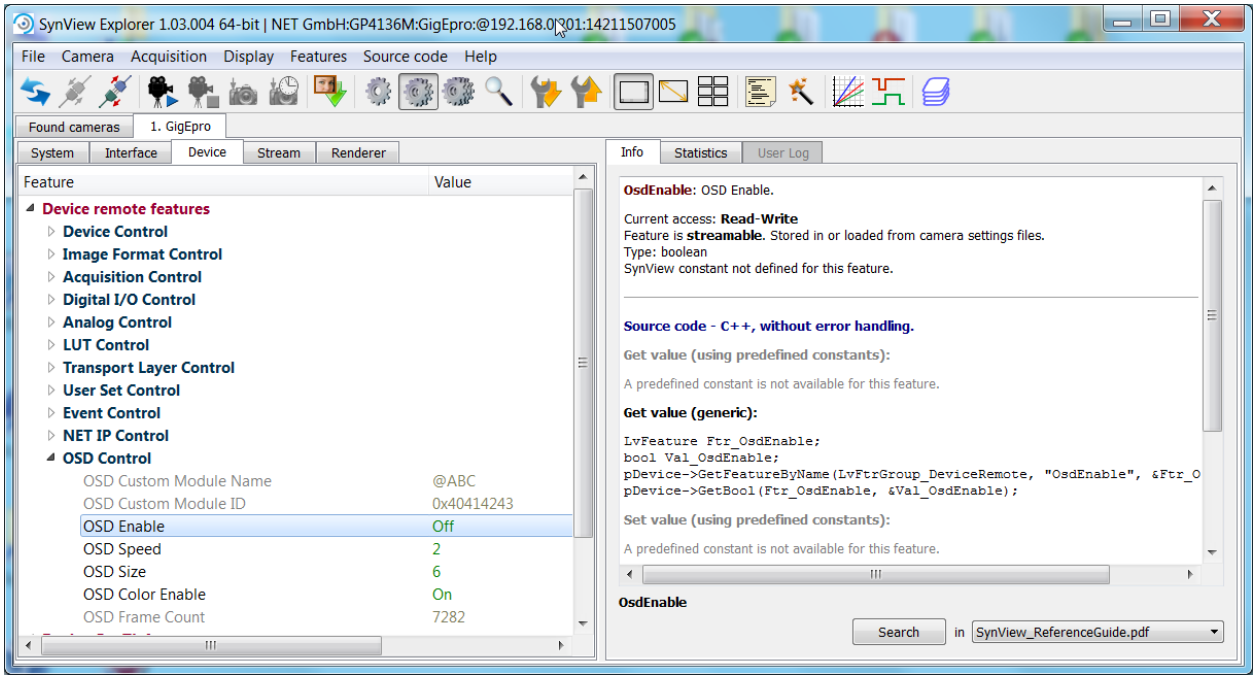

Figure 19: OSD control

XML Download

The customized XML category file has to be merged with the standard GigEPRO XML file with the xmlPreProc tool. The final XML file can be transfer to the GigEPRO flash memory with the GigeFPGAFlasher tool. The required win32 tool can be found in the `/ODK/tools/bin` directory. For this the SynView SDK has to be installed on the PC.

The `/ODK/tools` folder contains batch files for the XML processing. The download tool has to identify the camera, therefore the actual IP address (or the model name) of the camera has to be listed in the batch file before execution. The final XML file can be checked by any XML verification tool before download, therefore the corresponding schema file can be found in the directory `/ODK/tools/data`. If the XML file is not GenICam compliant the camera can not connected again and the original XML has to be restored.

Hint: Make sure that the camera is not connected to any application before starting the download.

BlockDemo_XmlFlashing.bat:

```
rem *** download customized XML to GigEPRO ***

bin\xmlPreProc data/gigepro_0001.xml
../src/open_camera/blockDemo/blockDemo.xml temp/finalDownloadTemp.xml
bin\GigeFpgaFlasher 192.168.0.222 temp/finalDownloadTemp.xml xf
```

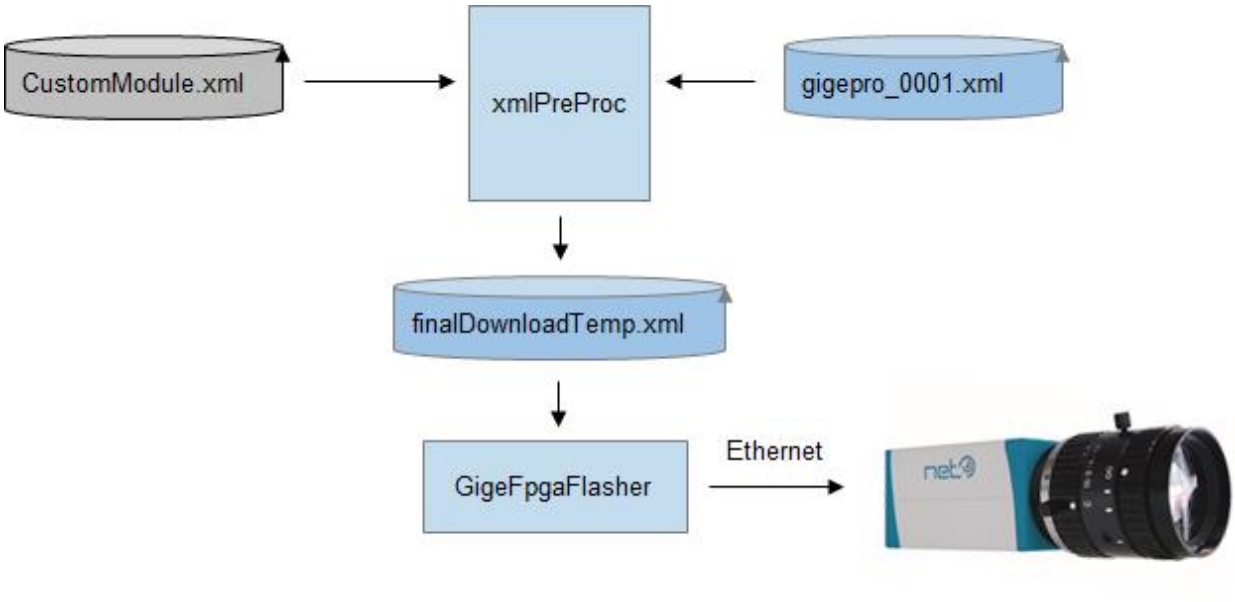

Figure 20: XML Download

Restoring GigEPRO Configuration

The customized GigEPRO Open Camera may be not connectable when a false FPGA configuration or XML file is downloaded to its flash memory. In this situation the original GigEPRO flash content can be restored with the Xilinx ISE Impact Tool and the `sv.ipconf.exe` tool from the SynView SDK.

FPGA Recovery

During the developement phase the Custom Module HDL design may disturb the normal camera operation, e.g. in case of “System Bus” deadlock. In this case the Xilinx ISE Impact tool and the Xilinx Platform Cable USB II have to be used. The original FPGA configuration file `bitstream_original_lx???.bit` can be found in the directory `/ODK/tools/data`. After reconfiguration the camera will reboot automatically. Now a valid FPGA configuration should be downloaded to the camera flash memory again (e.g. with the help of `sv.ipconf.exe`).

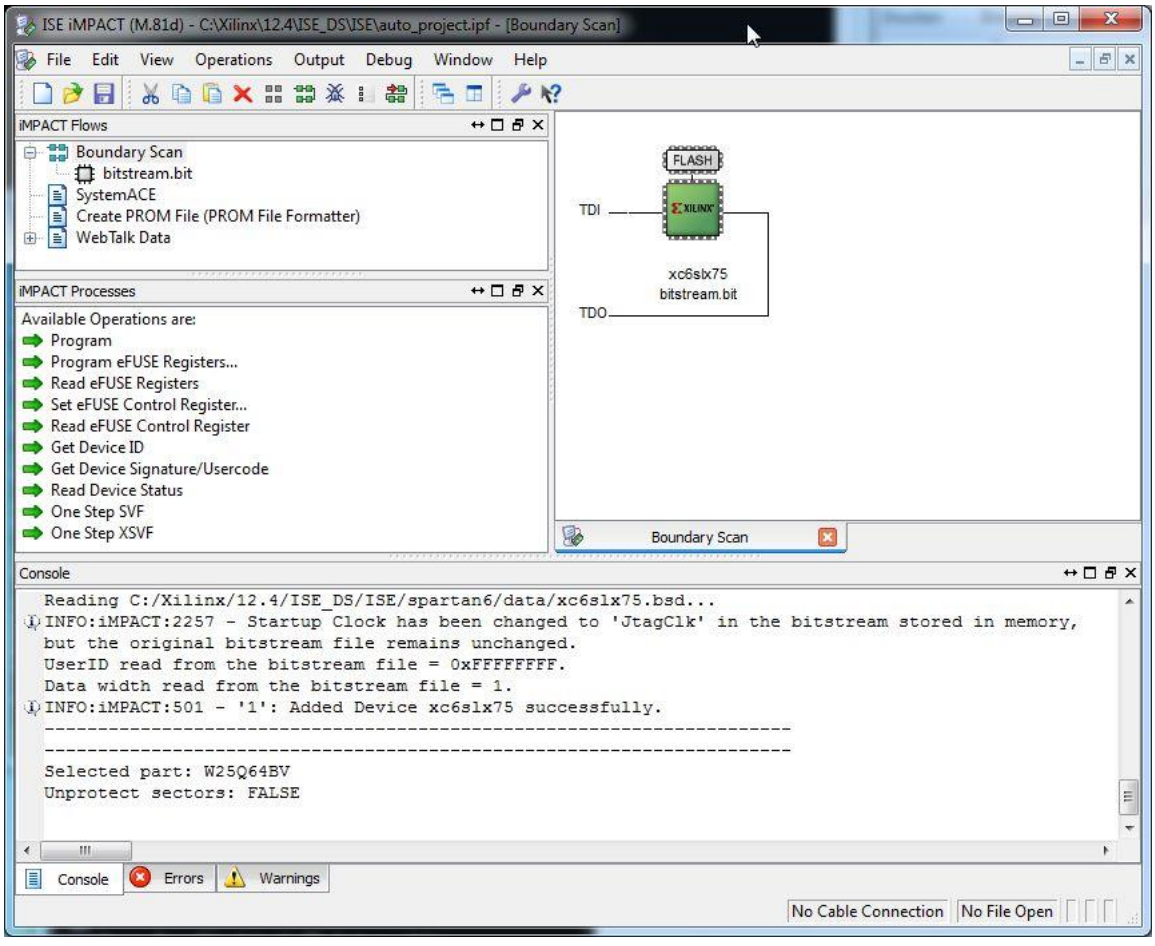

Figure 21: ISE Impact Tool

XML Recovery

When the camera XML file is not valid, the GigEPRO camera cannot be connected again. In this situation the GigeFpgaFlasher tool cannot access the camera. For XML recovery the `sv.ipconf.exe` tool from the SynView SDK has to be used. The tool has to be started with the –e option. The `/ODK/tools` folder holds a batch file for this operation (`ipconfRestoreTool.bat`). After selecting the camera the menu item “Update firmware mode” must be selected. The button “Update expert mode” opens a dialog, which allows XML file download to the camera.

Hint: Make sure that the camera is not connected to any application before starting the configuration.

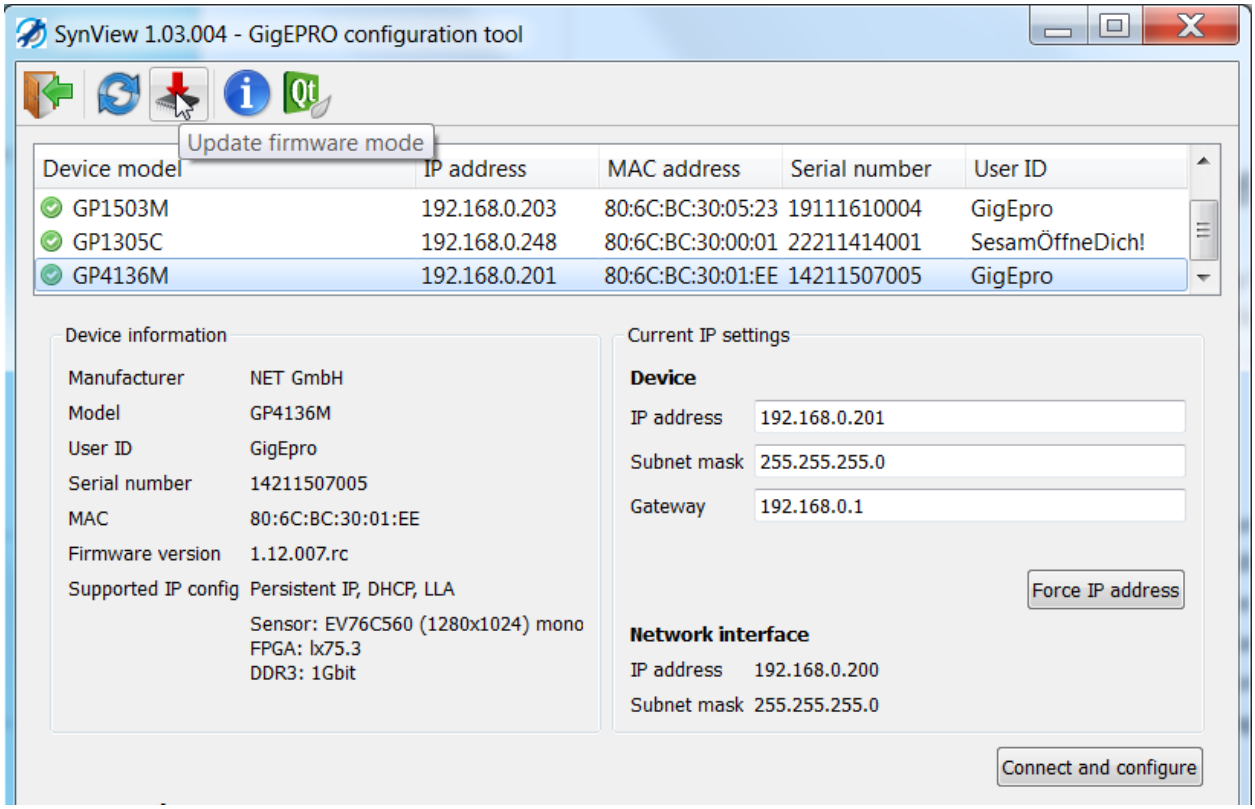

Figure 22: SynView sv.ipconf.exe Tool

Open Camera Data Flow

The Custom Module is integrated at the end of the image processing chain of the GigEPRO. It gets input pixel data from the Geometry Correction and delivers output pixel to the Frame Buffer controller. It operates as a streaming module without any flow control (i.e. continuous pixel stream at input and output port).

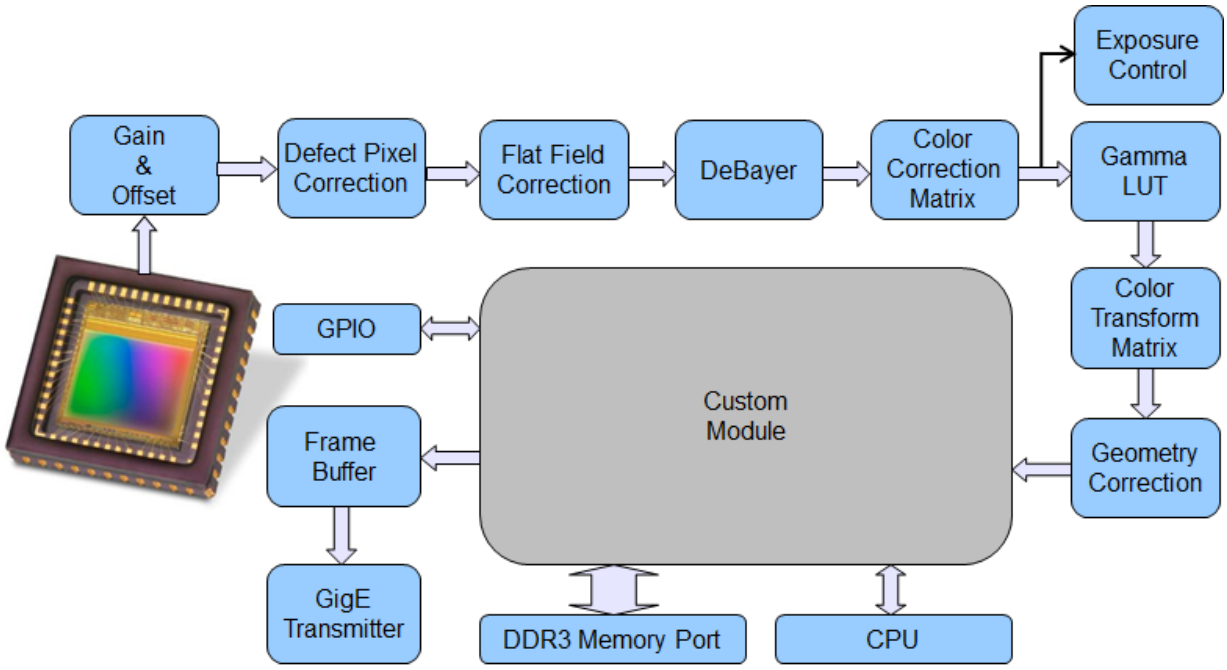

Figure 23: Open Camera Data Flow

A wrapper shell is build around the Custom Module to adapt the well specified interfaces to the internal GigEPRO design infrastructure. Part of this wrapper is a bypass function which is used to completely bypass the Custom Module. In this case the GigEPRO delivers the standard image stream to the host application. The Custom Module bypass has to be turned off by the application in order to get customized image processing.

The Frame Buffer controller maps the image data oputput (**ImgPixData[35:0]**) of the Custom Module according to the PixelFormat feature setting of the GigEPRO camera (see GigE-Vision 1.2 specification for pixel format definition). For example in case of “Mono8” pixel format, the camera will transmit only **ImgPixData[35:28]** as payload.

NET offers 3 different camera versions with predefined Custom Module:

- Canny
- Scaler
- HDR

There are also several Custom Module examples in the Open Camera Development Kit. Therefore the following Custom Module ID values must not be used by proprietary Custom Module designs:

Custom Module	ID (hex)	ID (ASCII)
Canny Edge Detector	0x636e6e79	cnny
DiffPic	0x4450494f	DPIO
BlockDemo	0x40414243	@ABC
Scaler	0x53636c44	SclD
HDR	0x48445231	HDR1

When the Canny ID is detected in the Custom Module ID register, the GigEPRO camera switches to a more user friendly XML interface. The same happens when a Scaler or HDR ID is detected.

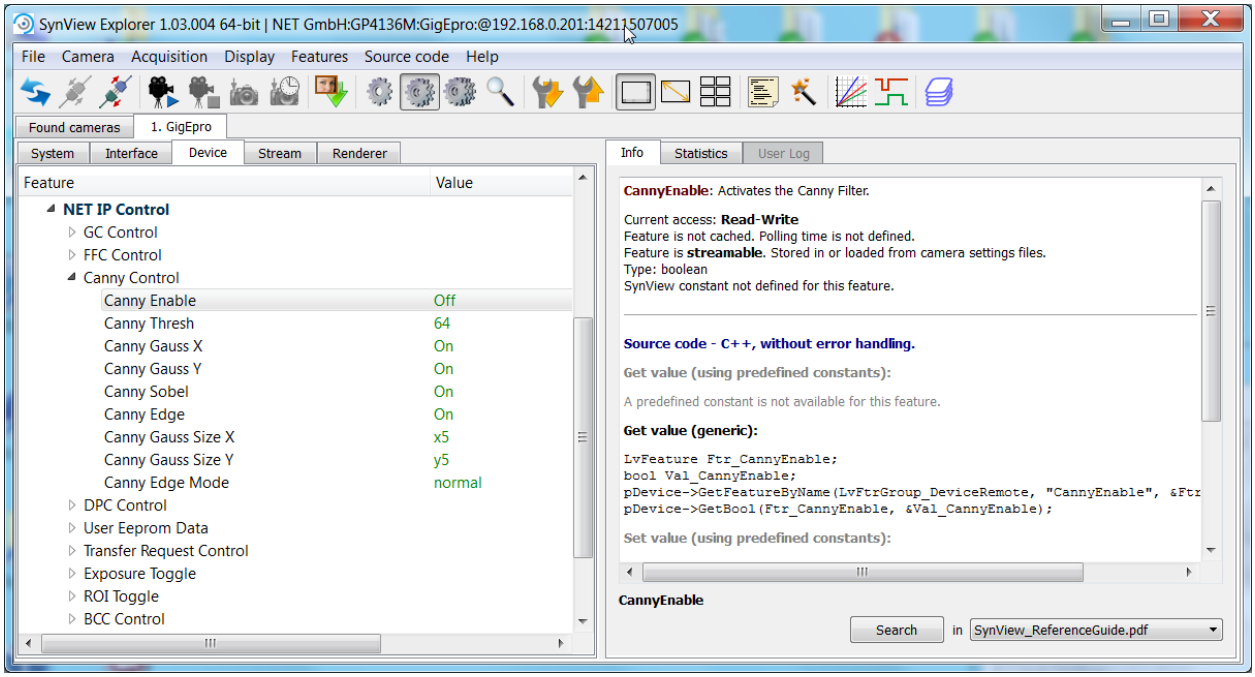

Figure 24: Canny Control

When a proprietary Custom Module ID is detected, the GigEPRO camera uses a generic XML interface. It consists of two parts:

- Custom Module Control
- Open Camera Control

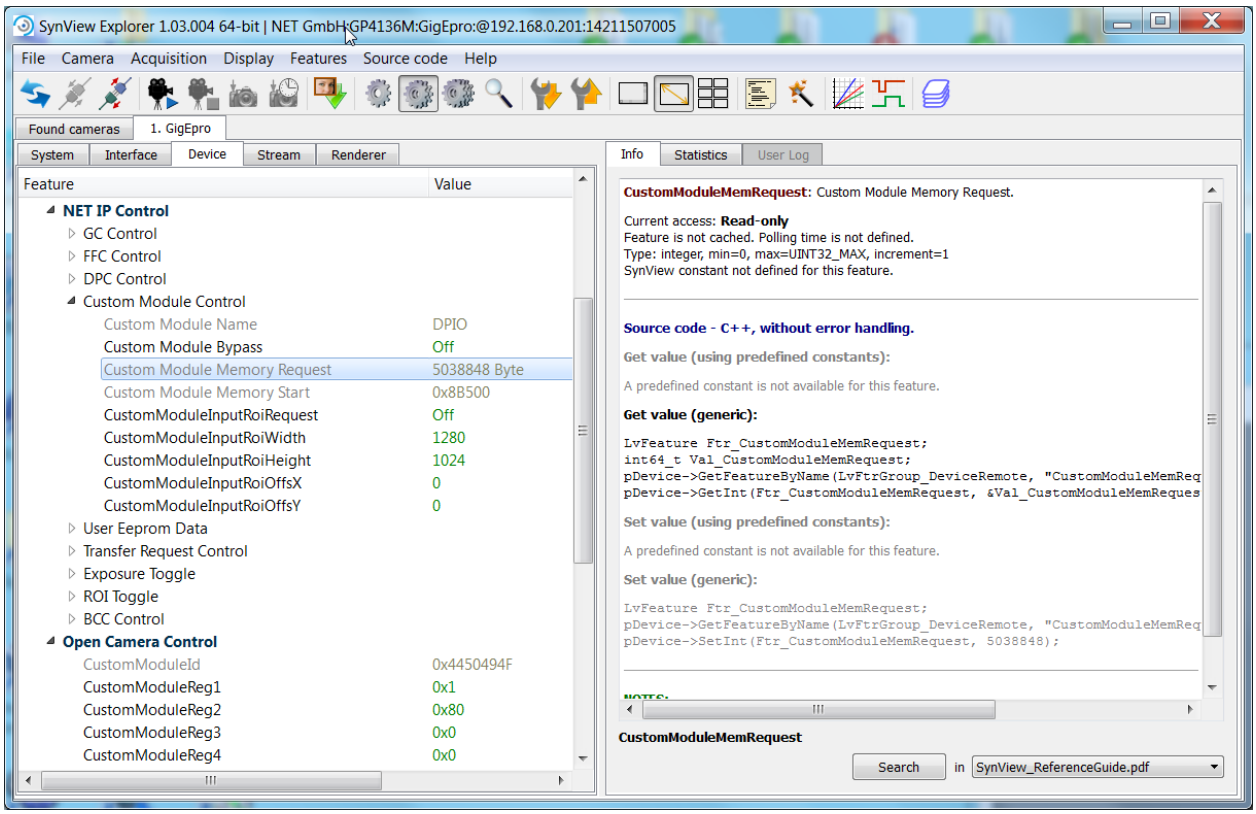

Figure 25: Custom Module Control

The Custom Module Control is used to control interaction between Custom Module and memory or sensor driver respectively:

- Custom Module Bypass
- Custom Module Memory Request
- Custom Module Input ROI Request

The “Memory Request” is used when the Custom Module needs access to the DDR memory of the GigEPRO camera. In this case the Custom Module design has to write the size of the desired memory buffer into the **MemSizeReq** register (0x7fc0). During boot phase the camera internal CPU will allocate the requested amount of memory and will write the start pointer into **MemStartAddr** register (0x7f80). Only after successful allocation of memory the CPU will set **MemWriteEn** register (0x7f81). The Custom Module must not access DDR memory before **MemWriteEn** register is set. The “Scale_D” example design in the Open Camera Development Kit demonstrates the implementation of DDR memory access.

The “Input ROI Request” is used when the Custom Module changes the size of the processed image (e.g. scaler). The size and format of the output image of the GigEPRO camera is defined by standard XML features “Width”, “Height” and “PixelFormat”. The Custom Module has to deliver exactly this image size. With the “Input ROI Request” feature it is possible to setup the sensor and the image chain in front of the Custom Module to process a different image size. If ”Input ROI Request” is disabled, the sensor and the image chain are operating on the same image size as the camera delivers to the appliation.

The Open Camera Control allows access to all registers inside the Custom Module.

IMPRINT

NET New Electronic Technology GmbH

Address:
Lerchenberg 7
D-86923 Finning
Germany

Contact:
Phone: +49-88 06-92 34-0
Fax: +49-88 06-92 34-77
www.net-gmbh.com
E-mail: info@net-gmbh.com

VAT- ID: DE 811948278
Register Court: Augsburg HRB 18494