GigE Vision Reference Design

for Transmitter Devices

Document Revision X-1.1.2, 2012-11-30

Sensor to Image GmbH
Lechtorstr. 20
D-86956 Schongau
Germany
http://www.sensor-to-image.de

Contents

Terms and Abbreviations

AOI	Area of Interest.
DDR	Double Data Rate interface. Signals of this interface might change its state on both rising and falling edges of a clock.
EPC	External Peripheral Controller core from Xilinx Platform Studio. It provides synchronous interface to connect custom peripherals to the CPU system bus.
GVCP	GigE Vision Control Protocol. See the GigE Vision Specification for details.
GVSP	GigE Vision Streaming Protocol. See the GigE Vision Specification for details.
MPMC	Multi Port Memory Controller core from Xilinx Platform Studio. It provides access to different types of memory through several ports.
SDR	Single Data Rate interface. Signals of this interface may change its state either on rising or on falling edge of a clock but not on both.
SDRAM	In this document it always means SDR Synchronous Dynamic RAM unless there is explicitly stated DDR or DDR2.
XST	Xilinx Synthesis Technology. The FPGA synthesis tool provided by Xilinx, Inc.

Overview

The GigE Vision Camera Reference Design is the official system for evaluation of the GigE IP core for transmitters and its companion cores named MPMC-Framebuffer core and Ethernet MAC core. The camera reference design is based on the CANCam-GigE/S3r2 main board and the CANCam-GigE Camera+CCD expansion board. In addition several Xilinx evaluation boards like the SP605 can be used. The design is done in Xilinx ISE version 13.4 and EDK version 13.4 and will be updated periodically.

Structure of the Design

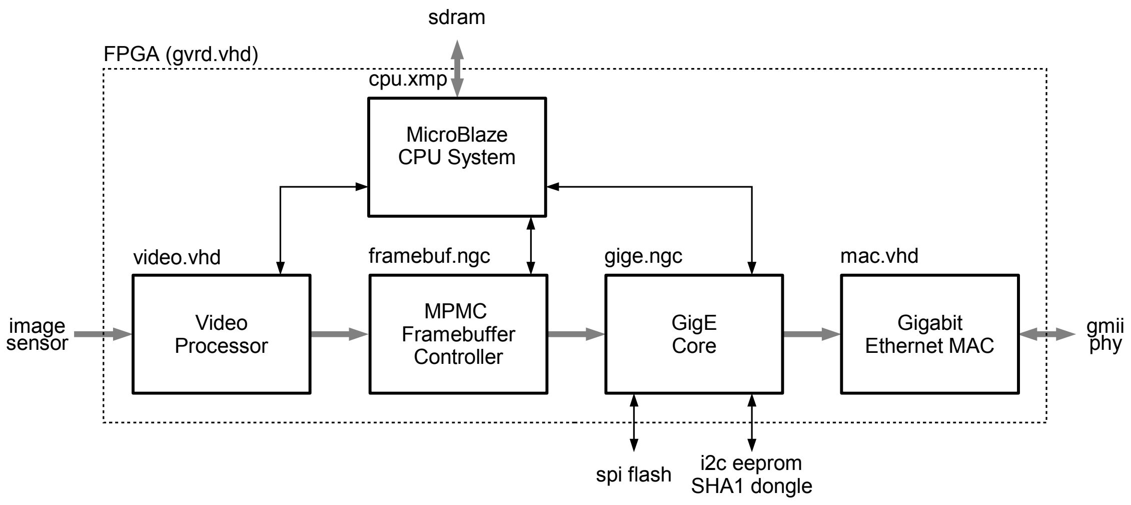

Figure 1: Block diagram of the camera reference design

The top level VHDL source file *gvrd.vhd* instantiates all the modules required to form fully functional GigE Vision camera. The design consists of following modules:

- Processor system *cpu.xmp*. This is an EDK generated processor system based on MicroBlaze CPU and PLB peripherals for interfacing the CPU to an external code memory and the rest of the reference design. It instantiates also Xilinx Multiport Memory Controller (MPMC), which provides access to external memory both from cpu and framebuffer core side.
  The reset generator forms global synchronous system reset based on external reset input and DCM locked status. The clock generator is based on the DCM. It generates system clock and sensor clock from external clock. Please check comments of reference design for exact frequencies.
- Video processor module *video.vhd*. This modules provides set of basic registers required for a GigE Vision camera. Additionally it converts video data stream from the image sensor to the format required by the memory controller. Reference designs for Xilinx evalboards may run without imager. Then this module is a simple image generator.
- GigE framebuffer controller *framebuf.ngc*. This module operates as a special video frame-buffer which forms data packets for the GigE core. It handles half of the packet resend

feature of the GigE Vision streaming protocol. It uses Xilinx MPMC core as memory controller, which is part of the processor system *cpu.xmp*. Note that physical memory is shared between µBlaze and framebuffer.

- GigE core *gige.ngc*. This module handles all the low-level networking features to the rest of the system. It forms the GigE Vision stream channel and provides networking interface for the CPU system. Additionally it handles half of the packet resend feature.
- Gigabit Ethernet MAC *mac.vhd*. This is just VHDL wrapper for the NGC netlist of a Gigabit MAC core or a hard macro if available (e.g. in Virtex devices).

Basic Operation

The reference design forms a basic GigE Vision camera. The streaming channel is handled completely in hardware and therefore it is capable to reach maximum data throughput of the Gigabit Ethernet network. The control protocol together with supporting features are handled by the MicroBlaze CPU in software. Basic video processing features are handled partially in hardware and partially directly by the image sensor.

Delivered Archive

The reference design is delivered as an archive containing the ISE project and corresponding firmware. Structure of the archive is following:

./cores

Directory containing the IP cores. It should contain *gige.ngc*, *framebuffer.ngc* and *mac.ngc* if no hard mac is used.

./cpu

The Xilinx MicroBlaze based CPU system used in the ISE project is placed into that directory.

./ise

Directory containing the Xilinx ISE project of the reference design, but not the vhdl source files

./sdk

This directory contains Xilinx SDK workspace with hardware specification, board support package and firmware project. It also includs source code of the firmware, static GigE library, binary file of the bootloader, and support files like Makefile, linker scripts and so on.

./src

Directory containing the Xilinx ISE project source files of the reference design, usually vhd and ucf files.

In following text, the root of the extracted archive will be always indicated as . (dot).

Hardware

When no Xilinx evaluation board is used, the reference design is based on the CANCam-GigE/S3r2 Main Board and the CANCam-GigE Camera+CCD expansion board. Basic characteristics of the reference design hardware are summarized in Table 1.

Parameter	Value	Units
Main board dimensions	69.85 × 50.8	mm
Expansion board dimensions	50.8 × 50.8	mm
Common supply voltage	8 – 12	V DC
Total power consumption	2.75	W
Ethernet interface	1000BASE-T	
Resolution of the image sensor	752 × 480	pixels
Color mode of the image sensor	Gray scale	

Table 1: Basic parameters of the reference design hardware

Main Board

The main board is standard product for interfacing any data source to the Gigabit Ethernet network. The board is assembled with following components:

- Power supply
- Two buffered LVCMOS inputs and two buffered LVCMOS outputs
- Pin header connectors for expansion board directly connected to 55 FPGA I/O pins
- JTAG connector
- RS-232 interface
- CAN interface
- 8 kB EEPROM for non-volatile storage of GigE parameters
- 8 MB SPI flash memory for FPGA configuration and CPU firmware
- 32 MB SDRAM with 32 bit data bus for video frame-buffer and CPU program memory
- Gigabit Ethernet PHY BCM5461 with RJ45 connector
- Xilinx Spartan-3ADSP FPGA XC3SD1800A-CSG484

Connectors

Top-view layout of the main board showing position and orientation of the connectors is shown in Figure 2. (J6 is usually not assembled)

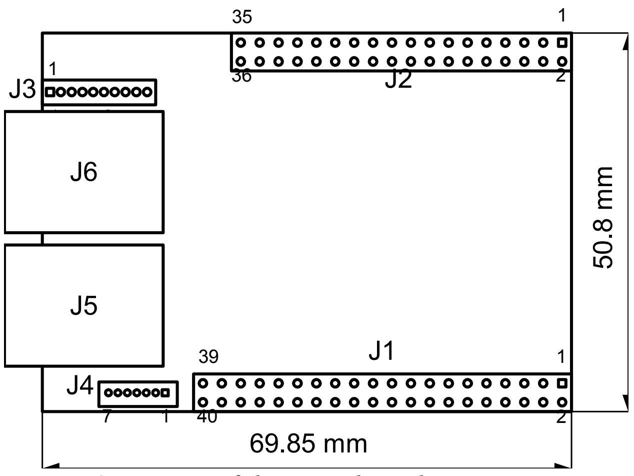

Figure 2: Layout of the main board

Position of the pin number 1 is always emphasized using a square shape in Figure 2. Numbering schema of the connector pins for top-view is described by Table 2.

2	4	6	...	N
1	3	5	...	N-1

Table 2: Numbering of the connector pins

The main board contains four pin header connectors. They are J1 and J2 for connecting an expansion board, J3 for power supply and general purpose I/Os, and J4 for JTAG connection to the FPGA.

Connection of the J3 connector is shown in Table 3. The J3 should remain unconnected in the reference design as there are power supply and RS232 connectors at the expansion board.

Signal	Pin Number		Signal
GND	1	2	Power supply input
LVCMOS input 1	3	4	LVCMOS input 2
Main board RS232 RXD	5	6	LVCMOS output 1
Main board RS232 TXD	7	8	LVCMOS output 2
Do not connect	9	10	Do not connect

Table 3: Pinning of the J3 connector

Connection of the JTAG connector J4 is shown in Table 4. Pin 6 is active-low input signal.

Pin Number	Signal
1	+3.3 V
2	JTAG TDI
3	JTAG TDO
4	JTAG TCK
5	JTAG TMS
6	Force FPGA reconfiguration
7	GND

Table 4: JTAG connector J4 pinning

Expansion Board

The expansion board was designed to present features of the main board and to allow simple evaluation of the GigE core. The expansion board contains following components:

- DC power supply circular jack 2.1 mm
- 9 pin d-sub connector for RS-232 interface
- Pin hole connectors to connect to the main board
- Flat ribbon cable connector to connect any Sensor to Image standard sensor board
- Aptina MT9V022 gray scale CMOS image sensor with resolution of 752×480 pixels

FPGA

The reference design itself is implemented in the Spartan-3ADSP FPGA. Block diagram is shown in Figure 1. Detailed description of the IP cores delivered as NGC netlists can be found in their respective documentation. The rest of the system is delivered as VHDL source codes to show example how to use the cores to create working GigE Vision system.

The reference design consumes around 50% of the XC3SD1800 slice resources. Table 5 shows precise numbers of device utilization. The values for the modules are estimated values coming from synthesis, which may vary slightly after full implementation.

Module	Slices	Slice FF	Slice LUTs	RAMB16s	DSP48As	DCMs
GigE core	-	3854	4823	15	0	0
Framebuffer core	-	6048	5814	4	1	0
1G MAC core	-	647	986	0	0	0
CPU System	-	3825	4716	13	3	2
Video processor and top level	-	410	384	1	0	2
Total after place and route	13032	14163	14679	33	4	4
% of XC3SD1800 resources	78%	42%	44%	39%	4%	50%

Table 5: Reference design FPGA resource utilization for XC3SD1800

Here the numbers for the design on SP605:

Module	Slices	Slice FF	Slice LUTs	RAMB16s	DSP48A1s	PLLs
GigE core	-	3542	4088	15	0	0
Framebuffer core	-	5522	4849	4	1	0
1G MAC core	-	695	731	0	0	0
CPU System	-	3026	3807	12	3	1
Video processor and top level	-	383	675	1	0	0
Total after place and route	4754	12491	11731	32	4	1
% of XC6SLXT45T resources	69%	22%	42%	27%	6%	25%

Table 6: Reference design FPGA resource utilization for XC6SLX45T

Firmware

The firmware is formed of C source code files and precompiled static library implementing all the functionality required by the GigE Vision specification. Structure of the firmware directory *./sdk/* is following:

Makefile	The makefile for the GNU make to control all build processes that create binaries and bitstream required to run the reference design.
bin/	The bin directory is created during make process and contains executables of the built application, bitstream created by merging current pure-hardware bitstream together with a bootloader, and binary image of the configuration EEPROM.
etc/	This directory contains supporting files like auxiliary Perl scripts, and GigE Vision device configuration XML file.
gvrd_s3adsp/	This directory contains C source files of the reference design application.
gvrd_s3adsp_bsp/	This directory contains board support package.
gvrd_s3adsp_hw/	This directory contains hardware description, which is used to generate board support package.
lib/	The GigE library *libgige.a* and the bootloader executable file *bootloader.elf* are placed into this directory.
src/	This directory contains C source files of the reference design application.

⚠ ***The Makefile can merge FPGA bitstream together with the bootloader and therefore it needs the bitstream to exist in the ./ise directory!***

SDK

Software development is done in the Eclipse based GUI called Xilinx Software Development kit. Three different projects are needed:

Hardware Platform Specification

This is the description of the EDK hardware design in an xml file. Every time EDK project is modified, the hardware design has to be exported to SDK, where it has to be imported in the hardware platform specification project. Name of this project in reference design is gvrd_s3adsp_hw or gvrd_sp605_hw for the SP605 reference design

Boards Support Package

Using the data of the hardware specification, this project defines the software layer for the final application project. It created drivers for the used hardware components. Name of this project in reference design is gvrd_s3adsp_bsp or gvrd_sp605_bsp for the SP605 reference design

Application

This project is the custom user application running on MicroBlaze. This projects refers to the libraries created in the board support package and also refers to a static library, which provides all gige vision related functions. Name of this project in reference design is gvrd_s3adsp or gvrd_sp605 for the SP605 reference design. Result of compile is an elf file either in debug or release mode, which is used by the makefile (see below) to create the application.bin file. Be sure to reference correct output elf-file in makefile.

Makefile

Final build process of the application as well as merging FPGA bitstream with bootloader is under control of the GNU make tool. Following section is for Windows users not familiar with UNIX shell and make utility. Users running Xilinx development tools under Linux or Solaris operating systems are usually used to use these tools.

Command Shell Environment and GNU Make

All the UNIX-like tools required for successful generation of the executable and bitstream are part of the Xilinx development environment under Windows operating systems. To enter the command line run the *ISE Design Suite Command Prompt* from the Accessories start menu or out of SDK. Now at the command line change current working directory to the sdk workspace directory. To change current working directory to *<ref-design-path>\sdk* on disk *D:* you must type following command:

```
d:

cd <ref-design-path>\sdk
```

When you are in this firmware directory, you can control build process of the firmware issuing following command:

```
make <target>
```

where the <target> is one of the Makefile targets described in following section.

Makefile Targets

The firmware *Makefile* offers following targets to control all required tasks of the build process:

all	This target generates or refreshes all the output files. It calls subsequently makefile targets *eeprom*, *bit and bin*.
bin	Creates raw binary file *application.bin* from the ELF file(created in SDK) and displays its length and checksum in hexadecimal.
bit	Merges the *gvrd.bit* FPGA bitstream from the ISE project together with *lib/bootloader.elf* creating *bitstream.bit* in the *bin* subdirectory. Then it converts the BIT file to raw binary *bitstream.bin* and displays its length and checksum in hex.
eeprom	Generates binary image of the configuration EEPROM and displays its length and checksum in hex. The image can be used to set default parameter values.
clean	Deletes all generated files.
prog	Programs *bin/bitstream.bit* into the FPGA using Xilinx Impact and JTAG cable.
run	Envokes xmd debugger and downloads *bin/application.elf* to be executed by µBlaze. Xilinx Impact and JTAG cable is used.

Bootloader

The bootloader is delivered as an ELF file together with the firmware. It is merged together with the FPGA bitstream during build process and therefore it is available always whenever FPGA is configured. Its basic function is to load the firmware from SPI flash into program memory and start it. Additionally it allows to upload binary files from a PC using serial line and program these files into the flash memory or parameters EEPROM.

The bootloader communicates with the PC using serial line set to following parameters:

Speed:	115200 Bd
Data bits:	8
Parity:	none
Stop bits:	1
Flow control:	none

After successful configuration of the FPGA using JTAG or from SPI flash the bootloader displays some version information and waits for 3 seconds printing out dots. When no key is pressed during this period, the bootloader tries to load a firmware from flash and if it is successful, it starts it.

When a key is pressed during the initial timeout period or when there is an error while loading the firmware from flash, the bootloader enters its command line. User has a full access to all the features of the bootloader from the command line.

The bootloader signalizes its current status using two general-purpose outputs. These can be used by custom hardware. States of the bootloader are summarized in Table 7.

sys_gpo[1]	sys_gpo[0]	State
0	0	The bootloader is working
0	1	Boot failure, bootloader is entering command line
1	0	Bootloader is passing control over to the application
1	1	Undefined state

Table 7: Bootloader status

Commands

Command line mode of the bootloader allows user full control over the bootloader features. The commands are entered pressing one key only. The command interpreter is case-insensitive. List of available bootloader commands is shown in Table 8.

Key	Command	Description
U	Upload	Upload binary file using Xmodem protocol over a serial line to the program memory
D	Download[1]	Download content of the program memory using serial line
L	Load	Load data from flash into program memory
S	Save	Save data from program memory into flash
R	Run	Execute program from the program memory

Table 8: Bootloader commands

The *Load* and *Save* commands expect additional argument at the command line. These additional arguments are listed in Table 9. The *EEPROM* command needs confirmation – pressing the *Y* key as “yes” – before rewriting contents of the parameters EEPROM.

Key	Command	Description
B	Bitstream	The command operates with the FPGA bitstream
S	Secondary Bitstream	Load or save secondary (backup) FPGA bitstream

1 The Download command is disabled in current version of the bootloader.

Key	Command	Description
X	XML File	Load or save an XML camera description file
A	Application	The command operates with the firmware application
E	EEPROM	Load or save contents of the parameters EEPROM

Table 9: Additional arguments of the Load and Save bootloader commands

The most important commands for system operation are *Upload*, *Load*, and *Save*.

Upload After issuing the *upload* command bootloader periodically tries to start the Xmodem file transfer over a serial line. Whenever the sending side (a PC) starts sending data, the bootloader stores it into the program memory. After end of the transfer the bootloader prints out length of the received file and its checksum. These values might be checked on a PC using the *sdk/etc/cksum.pl* Perl script.

Load This command expects one parameter and then it loads corresponding section of the flash memory or EEPROM into the program memory and prints out its length and checksum.

Save This command expects one parameter and according to it then stores the current contents of the program memory into the flash memory or EEPROM.

Due to different implementations of XMODEM protocol in TeraTerm, only version 3.1 is supported!

Boot Flash Memory

The bootloader uses an SPI flash memory for storage of the FPGA bitstreams, firmware, and device description XML file. Default memory map for S3ADSP devices of the flash memory is shown in Table 10, for SP605 in Table 11. Other reference designs on Xilinx eval boards may have different layout.

0x7F FFFF	0x50 0000	Application	3 MB
0x4F FFFF	0x41 0000	XML File	960 kB
0x40 FFFF	0x40 0000	Allocation Table	64 kB
0x3F FFFF	0x20 0000	Reserved for secondary Bitstream	2 MB
0x1F FFFF	0x00 0000	Bitstream	2 MB

Table 10: Flash memory map for S3ADSP board

For SP605 it looks like:

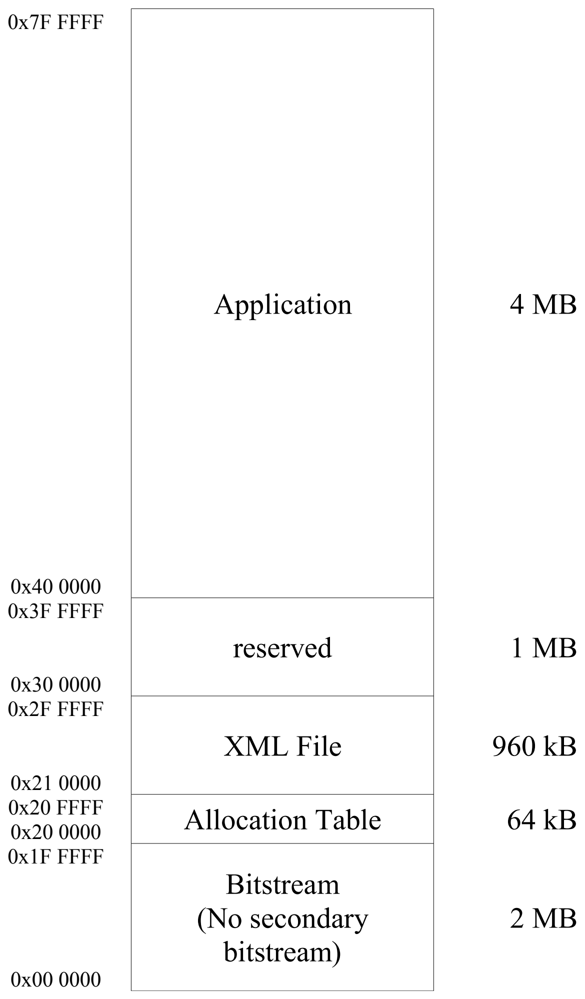

Table 11: Flash memory map for S3ADSP board

There is no special filesystem implemented in the flash memory. Its structure is rather fixed to keep the design simple. After power up the FPGA loads its configuration starting from address zero which should be a bitstream. The secondary bitstream could be used as backup when there would be a problem configuring the FPGA using the first bitstream (needs additional configuration).

The allocation table contains information about data stored in each section. It holds the length of the section in bytes and its checksum which is used during boot process to check data integrity. Table 12 shows occupation of the allocation table. Start address of each entry is only offset to absolute start address of table in flash.

0xFFFF : 0x0030	*Reserved*
0x002C	Application checksum
0x0028	Application length
0x0024	*Reserved*
0x0020	*Reserved*
0x001C	XML file checksum
0x0018	XML file length
0x0014	*Reserved*
0x0010	*Reserved*
0x000C	Secondary bitstream checksum
0x0008	Secondary bitstream length
0x0004	Bitstream checksum
0x0000	Bitstream length

Table 12: Layout of the flash memory allocation table

All values in the allocation table are 32 bit unsigned integer numbers. The length items mean number of used bytes within a corresponding section. The checksums are calculated as 32 bit sum of all used bytes of a section.

Parameters EEPROM

The I^{2}C EEPROM is used for non-volatile storage of constants and parameters required by the GigE Vision device. Detailed description of the EEPROM operation including its memory map can be found in the *GigE Vision IP Specification*.

There is a Perl script *eeprom.pl* in the *fw/etc* directory provided to allow user to prepare his own default settings of the parameters stored in the EEPROM. This script expects several parameters on its command line. These parameters are then used to assemble a binary image of the EPROM contents. The script can be executed from command line calling *xilperl etc/eeprom.pl [optional_parameters] -x <xml-file> -o <output-file>*. Table 13 shows command line options of the *eeprom.pl* script.

Option	Value	Description
-m (--mac)	hh:hh:hh:hh:hh:hh	Ethernet MAC address of the device
-c (--ipcfg)	d8	Enabled IP configuration methods, allowed values are 1 for static, 2 for DHCP, 4 for LLA, and all their “or” combinations
-i (--ip)	d8.d8.d8.d8	Static IP address of the device
-n (--net)	d8.d8.d8.d8	Static IP network mask
-g (--gw)	d8.d8.d8.d8	Static IP default gateway
-p (--port)	d16	GVCP port number
-u (--uname)	text	User defined name, maximum length is 15 characters
-d (--dest)	d8.d8.d8.d8	Destination IP address for bidirectional version
-s (--stream)	d16	GVSP port number for bidirectional version
-e (--serial)	text	Serial number/ID of the device
-t (--text)		Force the script to generate C source file containing initialized array instead of EEPROM binary image
-x (--xml)	text	Path to existing device description XML file
-a (--addr)	hhhhhhhh	Address of the XML file within the GigE Vision register address space
-o (--output)	text	Name of the output C source or binary image file

Table 13: Command line options of the eeprom.pl script

The command line arguments are expected in form “*option value*”. The *-x* and *-o* options are mandatory and must be always present. The other options are optional and predefined default values will be used when options will be omitted.

The values in Table 13 use simple symbols to represent actual parameters. The *h* represents a single hexadecimal digit, *d8* means 8-bit unsigned decimal number, *d16* stands for 16-bit unsigned decimal number, and *text* means a text string. When it is necessary to use a whitespace character within a text, enclose the whole text string into quotation marks.

Standalone Operation

To make the whole system run standalone, it is necessary to follow these steps:

- After any change in hardware generate up-to-date FPGA bitstream in Xilinx ISE. The ISE project must be located in the *./ise* directory and the bitstream must be named *gvrd.bit*. (reference name could be changed in makefile)
- Generate the firmware application elf file in Xilinx SDK.
- Generate the firmware application binary and the FPGA bitstream merged with the bootloader calling *make all* or simply *make* from the *./sdk* directory. Generated files will be placed into *./sdk/bin*.
- Connect JTAG cable and power supply to the reference board and switch the supply on.

- Start your favorite serial terminal program and setup its serial line parameters according to the Bootloader section of this chapter.
- Configure the FPGA calling *make prog* or Xilinx Impact. In the serial terminal window press any key after bootloader reports its version and starts printing dots.
- In bootloader press the *U* key to upload the bitstream binary file to the program memory. The bootloader tries to start Xmodem file transfer now. Use a send file using Xmodem protocol feature of you serial terminal program to send the *./sdk/bin/bitstream.bin* file to the device. After the end of file is detected, the bootloader reports size and checksum of the received file. You can check it comparing with a result of the *xilperl etc/cksum.pl -x bin/bitstream.bin* command.
- Now you can save the bitstream to the flash memory pressing the *S* key followed by the *B* key what means "save bitstream".
- Upload the application binary the same way as the bitstream. Press the *U* key to call the upload command, send using Xmodem protocol the *./sdk/bin/application.bin* and check its length and checksum according to result of the *cksum.pl* script.
- Save the application into the flash pressing the *S* key followed by the *A* key what means "save application".
- Generate your default configuration EEPROM image calling *xilperl etc/eeprom.pl [options] -x etc/gvrd_v12.xml bin/eeprom.bin* (executed by *make eeprom*).
- Upload the EEPROM image pressing the *U* key and sending the *eeprom.bin* file using Xmodem protocol.
- Save the EEPROM contents pressing the *S* key followed by the *E* key what means "save eeprom". Be patient, programming whole EEPROM takes approximately 20 seconds.
- Upload the XML file using the *U* key and save it into flash memory using the *S* key followed by the *X* key what stands for "save XML".
- If everything passed smoothly, you can switch the power off and back on. You should see the bootloader information in the serial terminal, then several dots while waiting for boot, and finally it should report boot information and start the application.

User Application and GigE Library

The sample application is delivered as C source codes with the GigE static library. The sources are placed in the *sdk/gvrd_s3adsp/src* directory. The user code is separated from the GigE Vision related code. The user application does not need to deal with the GigE Vision protocol. Instead it uses services of the GigE library. The library can be found in *sdk/lib/libgige.a*.

The GigE library handles all the networking functionality of the reference design except the GigE Vision Streaming Protocol. The GigE static library has been compiled using MicroBlaze GCC compiler which is part of Xilinx EDK.

Detailed description of the library can be found in the GigE Vision IP Specification.

⚠ ***When manually creating the application project, be sure to reference the libgige in linker settings.***

GigE Vision Registers

From the perspective of the GigE Vision application, the control of the GigE Vision device is done using set of registers. Table 14 shows the register address map as it is implemented in the reference design.

0xFFFF FFFF 0xFF80 0000	Access to Boot SPI Flash Memory	8 MB
0xFF7F FFFF 0xFF7F E000	Configuration EEPROM	8 kB
0xFF7F DFFF 0x0000 A000	Reference Design Control Registers	4087.953 MB
0x0000 9FFF 0x0000 0000	GigE Vision Bootstrap Registers	40 kB

Table 14: GigE Vision control register address space

The GigE Vision register address space is separated into four regions. The *GigE Vision Bootstrap Registers* region is specified in the GigE Vision Specification. See the specification for detailed information. The address space starting at address *0x0000A000* is dedicated to manufacturer-specific register space. This region is divided into three sub-regions in the reference design. One part is related to the reference design control registers while the top of the address space is dedicated to accessing the configuration EEPROM and boot SPI flash memories. Address map of the reference design control registers is described in following chapter.

Remote Access to the SPI Flash Memory

The reference design implements a way how to access the SPI flash memory remotely. This is useful when it is necessary to update firmware of a device which is connected to a network but is not directly reachable and therefore it is not possible to use bootloader as described above. Remote access to the SPI flash memory is provided using last 8 MB of the GigE Vision manufacturer-specific register address space.

Reading contents of the SPI flash memory is straightforward. The GigE Vision register address is directly translated into SPI memory address and four bytes starting at this translated address are read. The SPI address is

$$addr = reg - 0\mathrm{xFF800000}$$

where the *reg* is address of the GigE Vision register accessed by the GigE Vision application running at a PC. This allows the GigE Vision application to access whole content of the SPI configuration/boot flash memory.

Write access to the SPI flash memory is slightly more complicated than reading. There is no straight way to implement direct random write access. Writing data into the flash is interfered by 64 kB write buffer and an address register. See the address mapping in Table 15.

0xFFFF FFFF 0xFF80 0000	Reserved (write-protected)	8127.996 kB
0xFF81 0000	Write Address	4 B
0xFF80 FFFF 0xFF80 0000	Write Buffer	64 kB

Table 15: SPI flash memory write access registers

Write accesses to the SPI flash memory are organized in 64 kB blocks. To write a data block into the flash memory, the application must fill the *write buffer* with 64 kB of data. If smaller number of bytes than 64 kB is required to be written, the remaining space of the buffer should be filled with 0xFF bytes. When the buffer is ready to be written, the application must write starting SPI flash memory address into the *write address* register. The firmware then erases 64 kB block of the flash memory and programs it using data from the write buffer.

Closer description of the boot SPI flash memory can be found above in *Bootloader* section of this chapter.

Remote Access to the Configuration EEPROM

The reference design allows the GigE Vision application to remotely update contents of the configuration EEPROM. It is implemented using access to a dedicated address space region from *0xFF7FE000* to *0xFF7FFFFF*.

The EEPROM is accessed simply using read and write commands. The GigE Vision register address is directly translated into EEPROM address and four bytes starting at this translated address are read or written. The EEPROM address is

$$addr = reg - \mathrm{0xFF7FE000}$$

where the *reg* is address of the GigE Vision register accessed by the GigE Vision application running at a PC.

Access to the configuration EEPROM is not described in a device description XML file and therefore it is hidden for any standard GigE Vision application. This feature is supported by special software from Sensor to Image only.

The EEPROM image useful for the update is generated by the *eeprom.pl* script described above in this chapter.

Video Processor

The reference design comes with very simple video processor. It is the simplest image sensor video input block which might be used as a base for customer video processor. Figure 3 shows block diagram of the video processor unit. In case of a reference design without image, block *Intensity Correction* is replaced by a test pattern generator.

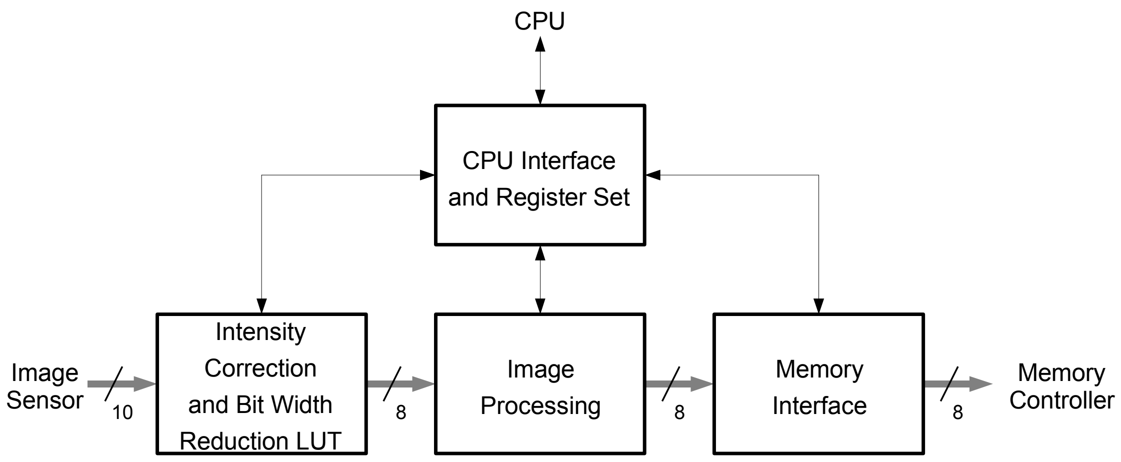
CPU
CPU Interface and Register Set
Image Sensor
10
Intensity Correction and Bit Width Reduction LUT
8
Image Processing
8
Memory Interface
8
Memory Controller

Figure 3: Block diagram of the video processor

The basic video processor unit corrects video signal gain using a look-up table. The look-up table also converts input 10 bit pixel data width to 8 bits. The 8 bit pixel data with corrected intensity can be used for video processing and then it is sent out to the memory controller. Interfacing to the CPU and basic video register set is provided using its dedicated block.

The image processing block is empty in the reference design. When additional image processing is required, this block must be replaced by the processing algorithm. This module must use input and output signals listed in Table 16. Direction of the signals is shown from perspective of the image processing module. The width is stated in bits.

Signal	Width	Dir.	Description
lut_frame	1	in	Frame valid from the image sensor
lut_line	1	in	Line valid from the image sensor
lut_strobe	1	in	LED strobe output of the sensor
lut_data	8	in	Pixel data from the look-up table
proc_frame	1	out	Processor frame valid
proc_line	1	out	Processor line valid
proc_data	8	out	Processed pixel data

Table 16: LUT output signals useful for custom image processing

CPU Interface

The CPU interface contains set of registers required for basic GigE Vision camera. Additionally it allows access to the input look-up table from the CPU software. The example firmware provides a way how to access these registers from a GigE Vision application running at a PC. It maps the video registers to the manufacturer-specific register space of the GigE Vision application allowing the application accessing them. Table 17 shows list of video registers provided by the reference design. In case of pattern generator, this varies slightly.

GigE Application		**Firmware**		**Name**	**Bits**	**Description**
Address	**Acc.**	**Offset**	**Acc.**			
0x0000A000	r/w	0x0000	r/w	gcsr	31..0	Global video control and status register; see below for details
0x0000A004	r/w	0x0004	r/w	width	11..0	Image frame width in pixels; allowed values are from 1 to max_width
0x0000A008	r/w	0x0008	r/w	height	11..0	Image frame height in lines; allowed values are from 1 to max_height
0x0000A00C	r/w	0x000C	r/w	offs_x	11..0	Horizontal offset of the AOI; it must be in range from 0 to (max_width - 1)
0x0000A010	r/w	0x0010	r/w	offs_y	11..0	Vertical offset of the AOI; it must be in range from 0 to (max_height - 1)
0x0000A014	r	0x0014	r/w	padding	31..0	Line and frame padding in bytes; see below for details
0x0000A018	r	0x0018	r/w	pixfmt	31..0	Pixel format according to the GigE Vision Specification
0x0000A01C	r	–	–	max_width	31..0	Maximum image width of the image sensor
0x0000A020	r	–	–	max_height	31..0	Maximum image height of the image sensor
0x0000A024	r	–	–	total_bpf	31..0	Total number of data bytes transferred for each image frame
0x0000A028	r/w	–	–	acq_mode	31..0	Current acquisition mode; the reference design supports continuous mode (1) only
0x0000A02C : 0x000197FF		–	–			Not used
0x00019800 : 0x00019FFF	r/w	–	–	lut	512 × 31..0	Video input look-up table; only first 256 dwords are used for the LUT

Table 17: Video processor CPU registers

There are two address and access type columns in Table 17. The *GigE Application* columns are related to the GigE Vision application running at a PC. The *Firmware* columns are valid for the firmware running in the embedded CPU of the reference design. The firmware offset means relative offset of the register address according to the EPC interface base address. Actual address of a video register is then

$$address = base + offset$$

where the *base* is typically *XPAR_EPC_x_PRHy_BASEADDR* defined in the *xparameters.h* header file. The *EPC_x* part is the name of the EPC instance in Xilinx Platform Studio and the *y* part is actual video processor peripheral number.

GCSR Register Bits

This is the global video control and status register. Description of its particular bits is shown in Table 18.

Bit(s)	Name	Access	Default	Description
31	acquisition	r/w	0	Video acquisition control; setting this bit to 1 starts continuous video acquisition, resetting it to 0 stops the acquisition
30..1				Not used
0	endianness	r/w	0	LUT access data endianness; 0 means big-endian, 1 stands for little-endian

Table 18: Video gcsr register bits

Padding Register Bits

This register sets the padding information of the GVSP leader packet. It is read-only for the GigE Vision application. Description of its particular bits is shown in Table 19.

Bit(s)	Name	Access	Default	Description
31..16	line_pad	r/w	0x0000	Number of insignificant bytes placed at the end of each video line as padding
15..0	frame_pad	r/w	0x0000	Number of insignificant bytes placed at the end of each video frame as padding

Table 19: Video padding register bits

The reference design does not need line padding and therefore the *line_pad* is kept zero. The *frame_pad* value depends on number of pixels per frame. Reference design firmware has a function to calculate correct padding values, dependent on framebuffer mode.

Revision History

Date	Version	Revision
2008-03-17	0.0.1	Initial draft of the document
2008-03-20	0.0.2	Extended draft
2008-03-26	0.0.3	Added description of the video processor
2008-04-01	0.0.4	Added flash memory map, description of the parameters EEPROM, and extended list of the bootloader commands
2008-04-02	0.0.5	Hyperlinks in table of contents
2008-04-09	0.0.6	Added GigE Vision registers section, updated resource consumption table
2008-04-15	0.0.7	Changes related to completely different eeprom.pl script
2008-04-24	0.0.8	Updated FPGA resources consumption
2008-05-05	0.0.9	Extended command line arguments of the eeprom.pl script
2008-05-22	0.0.10	Added bootloader "Evaluation" command
2009-01-23	0.1.0	Added remote access to the configuration EEPROM, new command line option of the eeprom.pl script, updated description of the delivered archive and firmware
2009-04-09	X-1.0	Final release of the Xilinx version
2009-05-18	X-1.0.1	Updated boot flash memory allocation table
2009-05-26	X-1.0.2	Added description of the bootloader status outputs
2009-07-23	X-1.0.3	Corrected specification of the supply voltage range
2009-12-02	X-1.0.4	Few minor corrections and modifications
2011-03-28	X-1.1.0	Update to MPMC based reference design
2012-11-27	X-1.1.1	Update for S3ADSP/SP605 Board Referencedesign (ISE12)
2012-11-30	X-1.1.2	Update for ISE13/14 with SDK for software develoment