GigE Vision Reference Design

for Receiver Devices

Document Revision X-1.1.0, 2013-02-27

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 Receiver Reference Design is the official system for evaluation of the stream channel receiver feature of the GigE IP core. The reference design is based on the Xilinx SP605 board together with S2I FMC-Sensor-Adapter V1.1. The design is done in Xilinx ISE version 14.4 and EDK version 14.4.

Structure of the Design

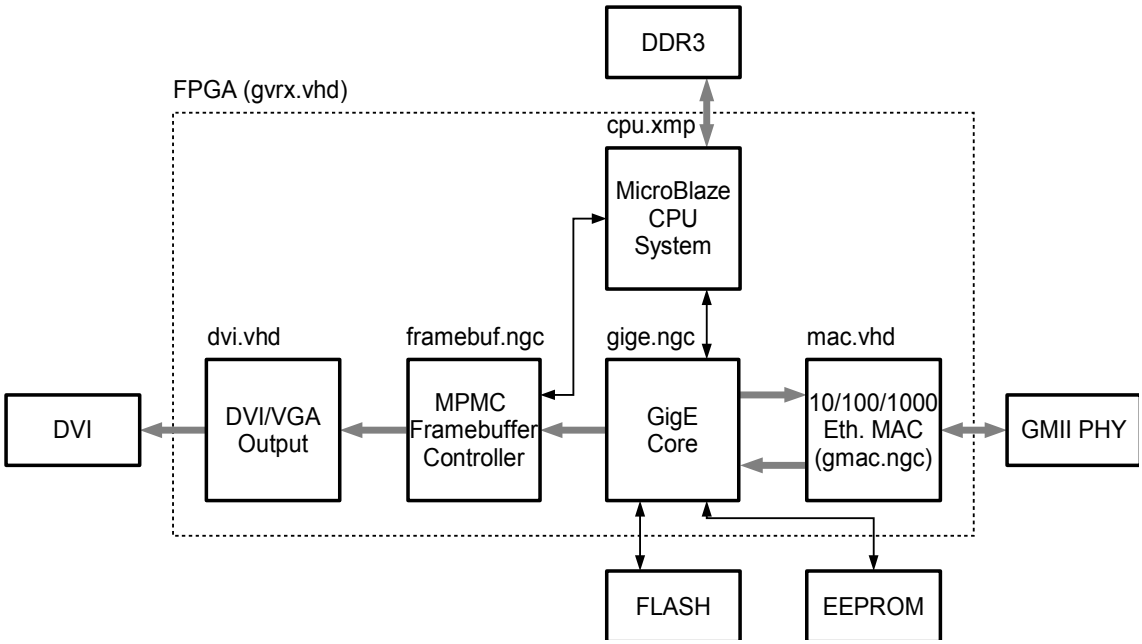

Figure 1: Block diagram of the receiver reference design

The top level VHDL source file *gvrx.vhd* instantiates all the modules required to form a system capable of receiving data from GigE Vision cameras and displaying it on attached DVI monitor as a live video stream. The design consists of following modules:

- Processor system *cpu.xmp*. This is an EDK 14.4 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.
- Gigabit Ethernet MAC *mac.vhd*. This is just a VHDL wrapper for the NGC netlist of the Gigabit MAC core *gmac.ngc*.
- GigE core *gige.ngc*. This module handles all the low-level networking features. It forms the

receiver of the GigE Vision stream channel and provides networking interface for the CPU system.

- Video framebuffer *framebuf.ngc*. This controller stores data stream from its input into the external DRAM. The image frames are buffered in the memory and read out of the synchronously to the video output.
- DVI output module *dvi.vhd*. This modules generates output video timing and composes the image to be shown. The video window at the screen is filled by the data from the framebuffer controller.

Basic Operation

The receiver reference design forms a very basic system capable of displaying live video streamed by GigE Vision cameras over the Gigabit Ethernet. The receiver of the stream 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.

The GigE receiver reference design does following tasks after its power up:

- The device boots up, initializes its network connection, and starts discovering GigE Vision devices available within the LAN.
- When more than one device is discovered, the user has to select one of them in a serial console. When there is just one device available, it is selected automatically.
- The receiver connects to the selected GigE Vision device, sets up its control and streaming channel and reads out its XML device description file. The receiver requires the XML file to be stored locally inside the GigE Vision device with no compression.
- The receiver parses the XML file and finds the right registers to set resolution and start constant streaming. If this fails these important registers have to be set manually in code. See separate HowTo document for more information.
- The receiver forces the selected GigE Vision device to start streaming. Contents of the streaming channel is displayed at the attached DVI monitor.
- It is possible to restart device discovery from the serial console.

Delivered Archive

The reference design is delivered as a ZIP archive with name *rx_sp605.zip*. Contents of the archive is following:

./cores	Directory containing the IP cores. It should contain *gige.ngc, framebuf.ngc* and *mac.ngc*.
./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

The receiver reference design is based on the Xilinx SP605 board and the S2I FMC-Sensor-Adapter V.1.1. Please refer to the Xilinx documentation about a detailed hardware description.

Expansion Board

The FMC expansion board is needed for licensing and storing some configuration data. The expansion board contains following components:

- SHA1 dongle
- 8kByte EEPROM
- 16 General purpose IOs, directly connected to FPGA

FPGA

The reference design itself is implemented in the Spartan-6 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 code to show an example how to use the cores to form working GigE receiver system.

Table 2 shows resource utilization of the target FPGA XC6SLX45T.

Resource	Consumed	Total	%
Slice Registers	7999	54576	14
Slice LUTs	9559	27288	35
Slices	3423	6822	50
DCMs	1	8	12
PLL	2	4	50
BUFGMUXs	7	16	43
DSP48As	4	58	6
RAMB16s	35	116	30

Table 1: Receiver reference design FPGA resource utilization

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.
gvrx_sp605/	This directory contains C source files of the reference design application.
gvrx_sp605_bsp/	This directory contains board support package.
gvrr_sp605_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 gvrx_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 gvrx_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 gvrx_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 2.

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 2: 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 3.

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 3: Bootloader commands

The *Load* and *Save* commands expect additional argument at the command line. These additional arguments are listed in Table 4. 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 4: 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 SP605 in Error: Reference source not found.

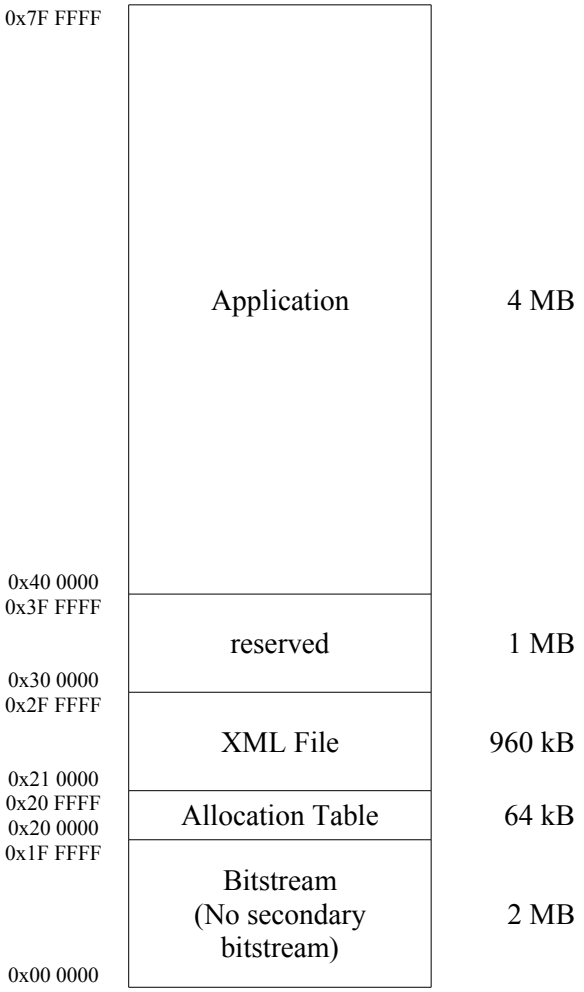

Table 5: Flash memory map for SP605 board

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 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. Error: Reference source not found 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 6: 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 I²C 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>*. Error: Reference source not found 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 7: 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 Error: Reference source not found 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 *gvrx.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_v10.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/gvrx_sp605/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. Error: Reference source not found shows the register address map as it is implemented in the reference design.

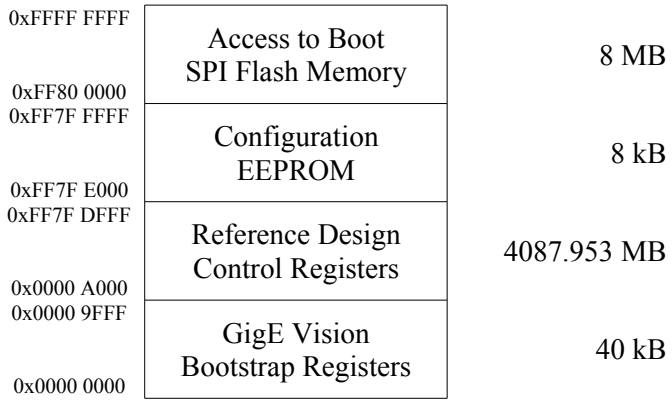

Table 8: GigE Vision control register address space

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

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

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 Error: Reference source not found.

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 9: 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

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

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.

These two blocks process the data received through the stream channel. The memory controller stores the data in an external DRAM memory forming single-buffered framebuffer. The DVI output displays the stored image on a DVI monitor.

Framebuffer

The framebuffer controller expects stream channel data at its input. Description of the interface signals including precise timing specification can be found in the *GigE Vision IP Specification* document.

The framebuffer controller consists of four VHDL source files. The *rdpim.vhd* and *wrpim.vhd* implement the read and write interface to the Xilinx MPMC-IP. The *fifo.vhd* forms an asynchronous FIFO. These modules are combined together in the framebuffer top-level file *framebuf.vhd*. Simplified block digram of the whole memory controller is shown in Figure 2.

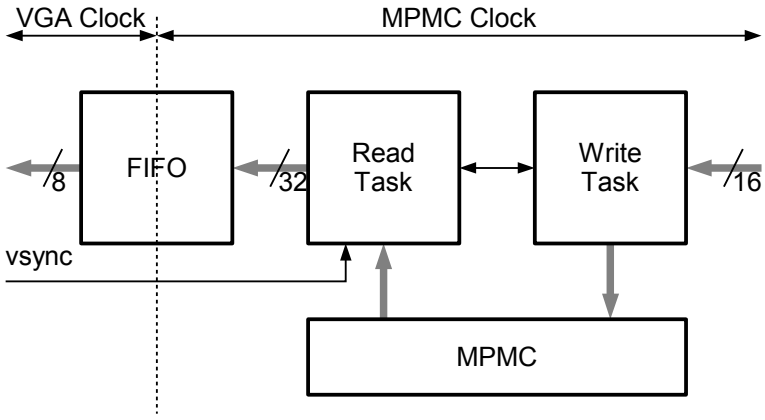

Figure 2: Block diagram of the framebuffer

The framebuffer controller stores incoming stream channel data into the buffer located in the external memory. Data width of the memory is 16 bits. The controller is formed of two main processes – one for writing incoming data into the MPMC and second for reading the data out.

The write task stores whole GVSP block into a single buffer. The task takes GVSP block ID and packet ID information into account so that each incoming packet of data is stored at its proper memory location.

The read task periodically reads contents of the buffer. The task is synchronized to the DVI output by its vertical synchronization signal. Whenever there is active level of the *vsync* signal, the read process resets its pointer to beginning of the memory buffer.

DVI Output

The DVI output module generates DVI timing and controls what will be displayed on the monitor. It can do potential Bayer-pattern to RGB color-space conversion if it is required. The

module generates signals for the 1280×1024 resolution at 60 Hz frame refresh rate. The screen is used according to Figure 3.

DVI chip on SP605 produces also VGA timing, so a VGA monitor could be used as well.

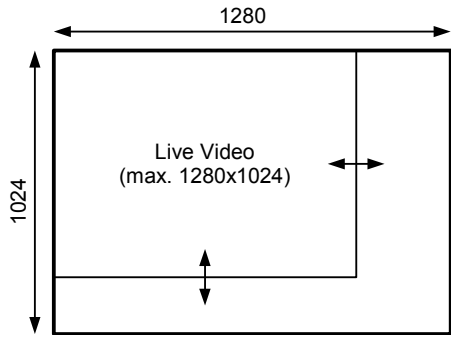

Figure 3: Arrangement of the monitor screen

The video stream received from the attached GigE Vision camera is displayed from the top-left corner of the screen. Resolution of the video is limited to 1280×1024 pixels. When actual resolution is smaller, the rest of the screen is blanked.

Revision History

Date	Version	Revision
2008-10-23	0.0.1	Initial draft of the document
2008-11-07	0.0.2	Few corrections and extension, added “Framebuffer and VGA Output” chapter
2009-04-09	X-1.0	Official release of the Xilinx version
2009-07-23	X-1.0.1	Corrected specification of the supply voltage range
2009-12-02	X-1.0.2	Resolutions up to 1024×768 pixels and Bayer-pattern color streams are now supported
2013-02-28	X-1.1.0	Update for 14 with SDK for software develoment. Adaption for SP605