GigE Vision Reference Design

for Transmitter Devices on
MVDK + Enclustra SA1 + FMC Sensor Adapter

Document Revision A-1.0.0, 2016-12-22
Contents

Disclaimer/Support.................................................................4
Terms and Abbreviations.....................................................5
Overview..................................................................................6
  Structure of the Design.......................................................6
  Basic Operation....................................................................7
  Delivered Archive..............................................................7
Hardware................................................................................9
  FPGA.....................................................................................9
  Core Licensing....................................................................9
  Core Update.........................................................................9
Firmware...............................................................................11
  ARM DS-5............................................................................11
    Application........................................................................11
  Preloader...........................................................................11
    Preloader Generation.......................................................12
  Makefile............................................................................12
    Command Shell Environment and GNU Make.....................13
    Makefile Targets...........................................................13
    Boot Flash Memory.........................................................13
    Parameters EEPROM.....................................................15
Standalone Operation..........................................................15
  User Application and GigE Library.....................................16
  GigE Vision Registers.......................................................18
    Remote Access to the SPI Flash Memory..........................18
    Remote Access to the Configuration EEPROM................19
Video Processor.....................................................................20
  CPU Interface.....................................................................21
    GCSR Register Bits........................................................22
    Padding Register Bits......................................................22
Disclaimer/Support

The information disclosed to you hereunder (“the Materials”) is made available “AS IS” and with all faults. Sensor to Image hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE. Sensor to Image shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Sensor to Image had been advised of the possibility of the same. Sensor to Image assumes no obligation to correct any errors contained in the Materials, or to advise you of any corrections or update. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent.

For free designs, support is limited to official documentation only.
For additional support, please contact:

Sensor to Image GmbH
Lechtorstrasse 20
D-86956 Schongau
Phone: +49 8861 2369 0
Fax: +49 8861 2369 0
Mail: email@sensor-to-image.de

© 2016 Sensor to Image GmbH
Terms and Abbreviations

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOI</td>
<td>Area of Interest.</td>
</tr>
<tr>
<td>AXI</td>
<td>Advanced eXtensible Interface Bus.</td>
</tr>
<tr>
<td>DDR</td>
<td>Double Data Rate interface. Signals of this interface might change its state on both rising and falling edges of a clock.</td>
</tr>
<tr>
<td>DS-5</td>
<td>ARM Development Studio 5 Intel SoC FPGA edition</td>
</tr>
<tr>
<td>EDS</td>
<td>Intel SoC FPGA Embedded Design Suite.</td>
</tr>
<tr>
<td>EPC</td>
<td>External Peripheral Controller core from Xilinx Platform Studio. It provides a synchronous interface to connect custom peripherals to the CPU system bus.</td>
</tr>
<tr>
<td>FMC</td>
<td>FPGA Mezzanine Card.</td>
</tr>
<tr>
<td>FSBL</td>
<td>First Stage Bootloader</td>
</tr>
<tr>
<td>GVCP</td>
<td>GigE Vision Control Protocol. See the GigE Vision Specification for details.</td>
</tr>
<tr>
<td>HPS</td>
<td>Hard Processing System, the ARM CPU system of Intel SoC FPGA</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>MVDK</td>
<td>Machine Vision Development Kit</td>
</tr>
<tr>
<td>SDR</td>
<td>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.</td>
</tr>
<tr>
<td>SDRAM</td>
<td>In this document it always refers to SDR Synchronous Dynamic RAM unless there is explicitly stated DDR/DDR2/DDR3.</td>
</tr>
<tr>
<td>S2I</td>
<td>Sensor to Image GmbH</td>
</tr>
<tr>
<td>SoC</td>
<td>System on a chip</td>
</tr>
</tbody>
</table>
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 AXI-Framebuffer core and Ethernet MAC core. The camera reference design is based on the S2I Machine Vision Development Kit MVDK, Enclustra Mercury SA1 SoC module with the AR0134 FMC sensor adapter board. The design is done in Quartus Prime version 15.1 and will be updated periodically.

The standard design described in this document uses an Aptina AR0134 imager as video source.

Structure of the 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.qsys`. This is a Qsys generated processor system based on Cyclone V Hard Processor System. It forms a processor system based on Intel SoC ARM A9 CPU and AXI peripherals for interfacing the CPU to an external code memory and the rest of the reference design. It instantiates also the AXI Master Memory Controller, which provides access to external memory both from CPU and framebuffer core side. The clock generator is based on HPS PLL. It generates a basic system clock, network clock and sensor clock from the internal master clock. Please check the inline documentation of reference design source files for exact frequencies.

- Video processor module `video.vhd`. This module provides an interface to the external image

---

Figure 1: Block diagram of the camera reference design
sensor and a set of basic registers to control this module. Additionally it gives the option to reduce the 12bit video data stream from sensor to 8bit by skipping the 4 low significant bits. Finally it converts the video stream to the format required by the memory controller. This module has to be adapted when other sensors should be supported.

- **GigE framebuffer controller** `framebuf_0.qsys`. 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 (serving resend packets) of the GigE Vision streaming protocol. It uses an AXI interconnect to interface to the AXI Master memory controller, which is part of the processor system `cpu.qsys`. Note that physical memory is shared between ARM and the framebuffer. This component is delivered in encrypted source files and can be imported to the Quartus IP Catalog.

- **GigE core** `xgige_0.qsys`. 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 (decoding requests). This component is delivered in encrypted source files and can be imported to the Quartus IP Catalog. To interface the GigE to MAC core, the bridge IP core `xg2g.qsys` is used.

- **Gigabit Ethernet MAC** `mac.vhd`. This is just a VHDL wrapper for the Gigabit MAC core (`gmac_0.qsys`). This core component is delivered in encrypted source files and can be imported to the Quartus IP Catalog.

### Basic Operation

The reference design forms a basic GigE Vision 1.2/2.0 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 is handled by the ARM CPU in software.

### Delivered Archive

The reference design is delivered as an archive containing the Quartus Prime project and corresponding firmware.

- **/cores**
  Directory containing the IP cores as encrypted source files. It should contain `s2i_framebuf_<version>`, `s2i_xgige_<version>`, `s2i_xg2g_<version>`, `s2i_gmac_<version>`, core information text files and an Quartus license file for using S2I IPs. This directory works as additional IP search directory for Quartus IP Catalog.

- **/quartus**
  Directory contains the Quartus Prime project file `<proj>.qpf`, Quartus Settings File `<proj>.qsf` with pinning information and S2I IP index file `s2i-ip-cores.ipx`.

- **/quartus/s2i_ip**
  Directory contains the S2I IP cores used from Quartus IP Catalog. In the reference design it contains the cores for framebuffer, xgige, xg2g and gmac. This directory is created by Quartus Prime, so don't update any files here, but only in `<proj>/cores`. See chapter Core Update.

- **/src**
Directory contains the Quartus Prime project source and constraint files of the reference design, usually vhd and sdf files.

/cpu

The ARM based CPU system used in the Quartus Prime project is placed into that directory. Modifications should be done only from Qsys GUI.

/fw

This directory contains ARM DS5 Intel SoC FPGA edition workspace with firmware project. It also includes source code of the firmware, static GigE library, ARM Preloader and support files like Makefile, linker scripts and so on.
Hardware

The MVDK consists of an Enclustra Mercury SA1 SoC module, S2I machine vision development kit carrier board and S2I AR0134 FMC sensor adapter module. For further information on Mercury SA1 please refer to its documentation [1], for MVDK carrier board documentation see the hardware user guide for MVDK [2].

FPGA

The reference design itself is implemented on Cyclone V SoC FPGA. Block diagram is shown in Figure 1. Detailed description of the IP cores delivered as encrypted source files can be found in their respective documentation. The rest of the system is delivered as plain VHDL source code to show an example on how to use the cores to create a working GigE Vision system. The reference design consumes around 19% of the 5CSXFC6 slice resources. Table 1 shows precise numbers of device utilization. The values for the modules are estimated values coming from synthesis, which may vary slightly after full implementation.

<table>
<thead>
<tr>
<th>Module</th>
<th>ALMs</th>
<th>Registers</th>
<th>M10Ks</th>
<th>DSPs</th>
<th>PLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GigE core</td>
<td>2418</td>
<td>5284</td>
<td>32</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>XG2G core</td>
<td>174</td>
<td>583</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Framebuffer core</td>
<td>2883</td>
<td>5398</td>
<td>26</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Tri-Mode MAC core</td>
<td>400</td>
<td>704</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Total after Implementation</td>
<td>7768</td>
<td>14960</td>
<td>77</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>% of XC7Z015 resources</td>
<td>19%</td>
<td>18%</td>
<td>11%</td>
<td>1%</td>
<td>25%</td>
</tr>
</tbody>
</table>

Table 1: Reference design FPGA resource utilization for 5CSXFC6C6U23I7

Core Licensing

The S2I IP cores are delivered as licensed Quartus IP package. Each component exists of component definition file <core>_hw.tcl and reference encrypted VHDL files delivered in <proj>/cores directory. Quartus license file license_customer_all.dat is included in cores directory.

Delivered license file has to be referenced in License Setup of Quartus Prime to compile S2I IP cores and generate bitstream.

Core Update

The S2I IP cores are designed to be integrated to the Quart Prime IP Catalog. To ensure that the latest version of the IP cores are being used follow these steps:
- Delete the IP core from <proj>/cores directory.
- Copy the new core in cores directory.
- Create new IP index file. Please refer to chapter Makefile and Makefile Targets make ipx.
- Refresh the Quartus IP Catalog by clicking Refresh IP Catalog from IP Catalog settings box or by pressing button F5.
- The new core should now appear in Installed IP → Library → S2I IP Cores → S2I <core>
- Upgrade IP:
  - Open Core in project hierarchy with Qsys by double click.
  - Core will be initialized and an Upgrade IP Cores information message will appear.
  - Close information message and click Generate HDL to generate updated IP core.
- IP cores can also be upgraded automatically by launching Launch IP Upgrade Tool, when yellow box IP upgrade recommended in Project Navigator Window appears. Click Perform Automatic Upgrade to update selected IP cores.
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 /fw is as follows:

Makefile  The Makefile for the GNU make to control build process of eeprom image, binaries, bitstream and JTAG programming targets.

bin/  The bin directory is used to store flash and eeprom images. A physical EEPROM is available on the MVDK carrier. This directory contains the eeprom image, bitstream image, application image and Preloader image for QSPI boot.

etc/  This directory contains supporting files like auxiliary TCL scripts, GigE Vision device configuration XML file and Conversion Setup File (bitstream.cof) used to convert bitstream into desired format.

gvrd/  This directory contains C source files of the reference design application and Makefile to build bare metal project application.

lib/  The GigE library libgige.a is placed into this directory.

ARM DS-5

Software development is done in the Eclipse based GUI called ARM Development Studio 5 (DS-5) Intel SoC FPGA Edition. It is part of the Intel SoC FPGA Embedded Design Suite (EDS). Reference design software is delivered as bare metal application managed by Makefile.

Application

This project is the custom user application running on ARM. This project refers to the Intel SoC FPGA hardware libraries (HWLIB) and also refers to a static library, which provides all GigE Vision related functions. Name of this project in the reference design is gvrd. Output of the compile are an axf executable file and a binary file, that are need for debugging and application image generation.

⚠ Please note that for debugging ARM firmware an ARM DS-5 Intel SoC FPGA Edition license is needed. For generation of application binaries the license is not necessary.

⚠ Please note that the Eclipse based GUI will indicate some errors in the code. This is caused by missing include paths to Intel SoC FPGA specific files. These errors can be ignored as they are only marked by the GUI but the application is built by the Makefile. Please verify that the application gvrd.axf is built without errors reported in the console window.

Preloader

Intel SoC FPGA based systems use a second stage bootloader (SSBL), called “Preloader” to
setup basic functionality of the ARM system.
The Preloader application is based on the Intel SoC FPGA Preloader, extended by functions that configure the RGMII delays in the Mercury SA1 onboard Ethernet PHY, initialize the QSPI flash and release the USB and Ethernet PHYs from reset.

The changes mentioned above are done by adding Patch file 001_preloader.patch to Preloader directory before generation. The Patch file was initially done by Enclustra. The patch file needed for extended Preloader functionality is provided in directory <proj>/fw/etc.

**Preloader Generation**

- Start the SoC EDS Command Shell
- Start the BSP Editor from Command Shell
  
  `$ bsp-editor`

- In BSP Editor: File → New HPS BSP
- For Preloader settings directory select `<proj>/quartus/hps_isw_handoff/cpu_hps_0`
- Disable the Use default locations checkbox. Select `<proj>/fw/spl_bsp` for BSP target directory
- Hit OK
- Select boot source:
  - Enable BOOT_FROM_RAM check box to boot from JTAG
  - Enable BOOT_FROM_QSPI check box to boot from QSPI flash
    
    When booting from QSPI flash change QSPI_NEXT_BOOT_IMAGE to 0x80000
  - All other check boxes must be disabled
- Enable FAT_SUPPORT check box
  - Change FAT_LOAD_PAYLOAD_NAME to `gvrd-mkimage.bin`
- Hit Generate, wait for completion and hit Exit.
- Copy the Preloader patch file 001_preloader.patch into the Preloader directory `<proj>/fw/spl_bsp`
- In EDS command shell navigate to the Preloader directory `<proj>/fw/spl_bsp`
- Build the Preloader from generated Preloader sources. This may take a few minutes.
  
  `$ make`

- After generation was done successfully, two generated files are important for further processing:
  - `<proj>/fw/spl_bsp/preloader-mkimage.bin` is the Preloader flash image with the boot ROM required header for booting from QSPI flash.
  - `<proj>/fw/spl_bsp/uboot-socfpga/spl/u-boot-spl` is the Preloader elf file needed for running reference design with ARM DS-5 debugger.

**Makefile**

Additional task that are not handled by the Quartus Prime or Intel SoC FPGA EDS can be controlled using the GNU make tool. The following section is for Windows users not familiar with the UNIX shell and Make utility. Users running Intel SoC FPGA development tools under Linux or Solaris operating systems are usually familiar with 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 Intel SoC FPGA development environment under Windows operating systems. To enter the command line run SoC EDS Command Shell from Windows Start Menu. Now at the command line change current working directory to the firmware directory.

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:

- **eeprom** Generates binary EEPROM image and displays its length and checksum in hex. The image can be used to set default parameter values.
- **bit** Generates bitstream flash image *fpga.rbf.img.bin* using Quartus Prime tools in the *<proj>/fw/bin* sub directory. Please verify paths in *<proj>/fw/etc/bitstream.cof*.
- **app** Generates firmware application flash image *gvrd-mkimage.bin* in the *<proj>/fw/bin* sub directory.
- **img** Calls Makefile target *bit* and *app* for generation of bitstream and application flash images.
- **clean** Deletes all generated files.
- **prog** Programs *<proj>/quartus/output_files/gvrd-sal.sof* into PL using Quartus Programmer and the JTAG cable.
- **headers** Creates header files from Qsys system design with the bases addresses of all the IP modules residing in the FPGA fabric. Has to be called each time Qsys system changes and referenced components have changed.
- **ipx** Creates an IP index file (*<proj>/quartus/s2i-ip-cores.ipx*) for S2I IP cores, which are not located in an Quartus IP search directory per default. Without this index file S2I IP cores will not appear in Quartus IP Catalog. This target should be called after copying the project, whenever project paths change or when IP cores are updated.

⚠ If *Make* target *prog* is used, disable 'boot from QSPI' in order to work on the unprogrammed device.

**Boot Flash Memory**

The system uses a QSPI flash memory for storage of the Preloader, the FPGA bitstream, firmware and device description XML file. Default memory map for MVDK flash memory is shown in Table 2.
There is no special file system implemented in the flash memory. Its structure is rather fixed to keep the design simple but can be changed to customer's needs. After power up the FPGA loads its configuration starting from address zero which must be a valid Preloader.

Table 2: Default Flash memory map for MVDK board
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 TCL script `eeprom.tcl` in the `<proj>/fw/etc` directory provided to allow the user to prepare their 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 `tclsh etc/eeprom.tcl [optional_parameters] -x <xml-file> -o <output-file>`. Table 3 shows command line options of the `eeprom.tcl` script.

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

Table 3: Command line options of the `eeprom.tcl` script

The command line arguments are expected in the 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 3 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 Quartus Prime. The Quartus Prime project be located in the `<proj>/quartus` directory and the bitstream must be named `gvrd-sa1.sof`.
- Generate the firmware application axf file in ARM DS-5 Intel SoC FPGA Edition.
- Generate the Preloader application as described in chapter Preloader Generation.
- Generate bitstream and application flash images calling Makefile target `make img` from Embedded Command Shell. Generated files shall be placed into `<proj>/fw/bin`.
- Set boot mode to QSPI boot.
- Connect the JTAG cable and power supply to the reference board and switch the supply on.
- Programming of the QSPI flash via JTAG is done using Quartus HPS flash programmer utility. There are four Makefile targets to program QSPI flash according Boot Flash Memory Table shown Table 2.

⚠️ **Note that programming QSPI flash via JTAG can take very long!** In HPS Flash Programmer User Guide it is recommend not to be used fro programming large files because of low speed of operation! Fore example programming bistream flash image to QSPI flash takes more than one hour!

⚠️ Please verify path in Makefile target “progf_pre” is pointing to correct Preloader flash image and that Preloader was generated for booting from QSPI flash.

- Program Preloader flash image to the QSPI flash:
  ```
  make progf_pre
  ```
- Program application flash image to the QSPI flash:
  ```
  make progf_app
  ```
- Program bitstream flash image to the QSPI flash
  ```
  make progf_bit
  ```
- Program XML file to the QSPI flash:
  ```
  make progf_xml
  ```
- Start your favorite serial terminal program and setup its serial line parameters to 115kB, 8 databits, 1 stop bit, no flow control.
- After successful programming, apply power cycle and check terminal for correct network connection. You should be able to ping the device now.
- To update the EEPROM, start the GigE Updater utility, connect to the MVDK board, choose MVDK-SA1 platform, select EEPROM destination and click File to choose the `eeprom.bin` image (located in `<proj>/fw/bin`). Hit Update to store EEPROM image on the device.

⚠️ Please check document How To Run GigE Vision Reference Design on MVDK Board for a detailed description and how to speed up flash programming using S2I Update GigE Tool.

### 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 `<proj>/fw/gvrd/src-gvrd` and `<proj>/fw/gvrd/src-misc` directories. 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 `<proj>/fw/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 the ARM GCC compiler which is part of Intel SoC FPGA EDS.

Detailed description of the library can be found in the GigE Vision IP Specification.
⚠ When manually creating the application project in SDK, be sure to reference the libgige in application software Makefile.
GigE Vision Registers

From the perspective of the GigE Vision application the control of the GigE Vision device is done using a set of registers. Table 4 shows the register address map as it is implemented in the reference design.

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFEFF FFFF</td>
<td>Access to Boot SPI Flash Memory</td>
<td>16 MB</td>
</tr>
<tr>
<td>0xFE00 0000</td>
<td>Configuration EEPROM</td>
<td>8 kB</td>
</tr>
<tr>
<td>0xFBFF 0000</td>
<td>Reference Design Control Registers</td>
<td>4087.953 MB</td>
</tr>
<tr>
<td>0x0000 A000</td>
<td>GigE Vision Bootstrap Registers</td>
<td>40 kB</td>
</tr>
</tbody>
</table>

*Table 4: 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's control registers is described in following chapter.

Remote Access to the SPI Flash Memory

The reference design implements a method to access the SPI flash memory remotely. This is useful when it is necessary to update the firmware of a device which is connected to a network but is not directly reachable and therefore it is not possible to use JTAG programming as described above. Remote access to the SPI flash memory is provided using the last 32 MB of the GigE Vision manufacturer-specific register address space. (16MB used only for physical flash)

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

\[ \text{addr} = \text{reg} - 0xFE000000 \]

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 interfaced by a 64 kB write buffer and an address register. See the address mapping in Table 5.
Table 5: SPI flash memory write access registers

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF FFFF</td>
<td>Reserved (write-protected)</td>
<td>8127.996 kB</td>
</tr>
<tr>
<td>0xFE01 0001</td>
<td>Write Address</td>
<td>4 B</td>
</tr>
<tr>
<td>0xFE00 FFFF</td>
<td>Write Buffer</td>
<td>64 kB</td>
</tr>
<tr>
<td>0xFE00 0000</td>
<td>Write Address</td>
<td>4 B</td>
</tr>
</tbody>
</table>

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 the starting SPI flash memory address into the write address register. As equipped QSPI flash device on Mercury SA1 has 256 kB sectors, the firmware erases 256 kB block of the flash memory only when sector boundary will be written. Then it programs 64 kB block using data from the write buffer.

A more detailed description of the boot SPI flash memory can be found above in Boot Flash Memory section.

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 0xFBFF0000 to 0xFBFF1FFF.

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

\[ \text{addr} = \text{reg} - 0xFBFF0000 \]

where the \text{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 for the update is generated by the \text{eeprom.tcl} 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 2 shows block diagram of the video processor unit.

**Figure 2: Block diagram of the video processor**

The basic video processor performs AOI control to mask out invalid sensor data. It also may be configured to reduce pixel depth from 12bit to 8bit, which can be used for video processing. Finally image 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 6. Direction of the signals is shown from the perspective of the image processing module. The width is stated in bits.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Width</th>
<th>Dir.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vi fval</td>
<td>1</td>
<td>in</td>
<td>Frame valid from the image sensor</td>
</tr>
<tr>
<td>vi dv</td>
<td>1</td>
<td>in</td>
<td>Data valid from the image sensor</td>
</tr>
<tr>
<td>vi data</td>
<td>16</td>
<td>in</td>
<td>Pixel data from the image sensor</td>
</tr>
<tr>
<td>proc frame</td>
<td>1</td>
<td>out</td>
<td>Processor frame valid</td>
</tr>
<tr>
<td>proc line</td>
<td>1</td>
<td>out</td>
<td>Processor line/data valid</td>
</tr>
<tr>
<td>proc data</td>
<td>16</td>
<td>out</td>
<td>Processed pixel data</td>
</tr>
</tbody>
</table>

*Table 6: Signals useful for custom image processing*
The CPU interface contains a set of registers required for basic GigE Vision camera. The example firmware provides a way how to access these registers from a GigE Vision application running on a PC. It maps the video registers to the manufacturer-specific register space of the GigE Vision application allowing the application accessing them. Table 7 shows a list of video registers provided by the reference design.

<table>
<thead>
<tr>
<th>GigE Application Address</th>
<th>Firmware Access</th>
<th>Offset</th>
<th>Name</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000A000</td>
<td>r/w</td>
<td>0x0000</td>
<td>gcsr</td>
<td>31..0</td>
<td>Global video control and status register; see below for details</td>
</tr>
<tr>
<td>0x0000A004</td>
<td>r/w</td>
<td>0x0004</td>
<td>width</td>
<td>11..0</td>
<td>Image frame width in pixels; allowed values are from 1 to max_width</td>
</tr>
<tr>
<td>0x0000A008</td>
<td>r/w</td>
<td>0x0008</td>
<td>height</td>
<td>11..0</td>
<td>Image frame height in lines; allowed values are from 1 to max_height</td>
</tr>
<tr>
<td>0x0000A00C</td>
<td>r/w</td>
<td>0x000C</td>
<td>offs_x</td>
<td>11..0</td>
<td>Horizontal offset of the AOI; it must be in range from 0 to (max_width - 1)</td>
</tr>
<tr>
<td>0x0000A010</td>
<td>r/w</td>
<td>0x0010</td>
<td>offs_y</td>
<td>11..0</td>
<td>Vertical offset of the AOI; it must be in range from 0 to (max_height - 1)</td>
</tr>
<tr>
<td>0x0000A014</td>
<td>r</td>
<td>0x0014</td>
<td>padding</td>
<td>31..0</td>
<td>Line and frame padding in bytes; see below for details</td>
</tr>
<tr>
<td>0x0000A018</td>
<td>r</td>
<td>0x0018</td>
<td>pixfmt</td>
<td>31..0</td>
<td>Pixel format according to the GigE Vision Specification</td>
</tr>
<tr>
<td>0x0000A01C</td>
<td>r</td>
<td>–</td>
<td>max_width</td>
<td>31..0</td>
<td>Maximum image width of the image sensor</td>
</tr>
<tr>
<td>0x0000A020</td>
<td>r</td>
<td>–</td>
<td>max_height</td>
<td>31..0</td>
<td>Maximum image height of the image sensor</td>
</tr>
<tr>
<td>0x0000A024</td>
<td>r</td>
<td>–</td>
<td>total_bpf</td>
<td>31..0</td>
<td>Total number of data bytes transferred for each image frame</td>
</tr>
<tr>
<td>0x0000A028</td>
<td>r/w</td>
<td>–</td>
<td>acq_mode</td>
<td>31..0</td>
<td>Current acquisition mode; the reference design supports continuous mode (1) only</td>
</tr>
<tr>
<td>0x0000A02C</td>
<td>r/w</td>
<td>–</td>
<td>gain</td>
<td>7..0</td>
<td>AR134 gain</td>
</tr>
<tr>
<td>0x10000000</td>
<td>r</td>
<td>–</td>
<td>SCPS_MIN</td>
<td>31..0</td>
<td>Minimum stream channel packet size</td>
</tr>
<tr>
<td>0x10000004</td>
<td>r</td>
<td>–</td>
<td>SCPS_MAX</td>
<td>31..0</td>
<td>Maximum stream channel packet size</td>
</tr>
<tr>
<td>0x10000008</td>
<td>r</td>
<td>–</td>
<td>SCPS_INC</td>
<td>31..0</td>
<td>Stream channel packet size increment</td>
</tr>
</tbody>
</table>

Table 7: Video processor CPU registers

There are two addresses and access type columns in Table 7. 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 the relative offset of the register address according to the EPC interface base address. Actual address of a video register is then

\[ \text{address} = \text{base} + \text{offset} \]

where the base is typically HPS_0_ARM_A9_0_EPC_x_BASE defined in the cpu.h header file in `<proj>/fw/gvrd/qsys_headers` directory. This header file is not generated automatically. Makefile target `make headers` has to be called to generate Qsys header files. The `EPC_x` part is the name of the EPC instance in Qsys CPU block.

**GCSR Register Bits**

This is the global video control and status register. Description of its particular bits is shown in Table 8.

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Name</th>
<th>Access</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31..1</td>
<td></td>
<td></td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>0</td>
<td>acquisition</td>
<td>r/w</td>
<td>0</td>
<td>Video acquisition control; setting this bit to 1 starts continuous video acquisition, resetting it to 0 stops the acquisition</td>
</tr>
</tbody>
</table>

Table 8: 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 9.

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

Table 9: Video padding register bits

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-12-21</td>
<td>A-1.0.0</td>
<td>Initial Release</td>
</tr>
</tbody>
</table>

GigE Vision Reference Design, Document Revision A-1.0.0, 2016-12-22
Bibliography
