LogiCORE IP Image Edge Enhancement v3.0

Product Guide

PG003 October 19, 2011

XILINX®

Table of Contents

LogiCORE IP Image Edge Enhancement v3.0 Product Guide

Introduction

The Xilinx Image Edge Enhancement LogiCORE™ IP provides users with an easy-to-use IP block to enhance the edges of objects within each frame of video. The core provides a set of standard Sobel and Laplacian filters with programmable gain that adjust the strength of the edge enhancement effect.

Features

- Support for:
  - High-definition (1080p60) resolutions
  - Up to 4096 total pixels and 4096 total rows
- Programmable gain for edge directions
- Selectable processor interface:
  - EDK pCore
  - General Purpose Processor
  - Constant Interface
- Support for 8-, 10-, or 12-bit input and output precision
- YCrCb 444 input and output
- For use with Xilinx CORE Generator™ software 13.3 or later
- Xilinx Streaming Video Interface (XSVI) bus simplifies connecting to other video IP

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Spartan®-6, Virtex®-6, Virtex-7, Kintex™-7
Supported User Interfaces	General Processor Interface, EDK pCore AXI4-Lite, Constant Interface
Resources	See Table 1-1 through Table 1-4.
Provided with Core	
Documentation	Product Specification
Design Files	Netlists, EDK pCore files
Example Design	Not Provided
Test Bench	VHDL (2)
Constraints File	Not Provided
Simulation Models	VHDL, Verilog Structural Models and C Model (2)
Tested Design Tools	
Design Entry Tools	CORE Generator™ tool, Platform Studio (XPS)
Simulation(3)	Mentor Graphics ModelSim, Xilinx® ISim 13.3
Synthesis Tools	ISE 13.3
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.
2. HDL test bench and C Model available on the product page on Xilinx.com at http://www.xilinx.com/products/ipcenter/EF-DI-IMG-ENHANCE.htm
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

Overview

Overview

The edge enhancement core combines the outputs of Sobel and Laplacian operators with the original image to emphasize edge content as shown in Figure 1-1.

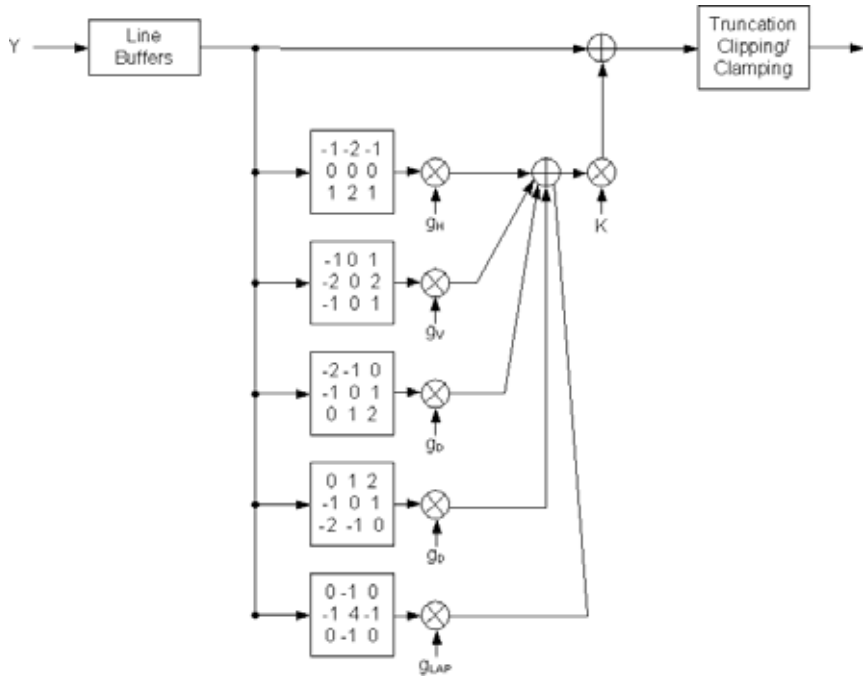

Figure 1-1: **Image Edge Enhancement**

Standards Compliance

AXI4-LITE Interface.

Feature Summary

The Image Edge Enhancement core uses Sobel and Laplacian filters to enhance edges of objects. There is a programmable gain for each filter to adjust the strength of the edge enhancement effect. This core works on YCbCr 4:4:4 data.

Applications

- Pre-processing Block for Image Sensors
- Video Surveillance
- Video Conferencing
- Video Capture Devices

Licensing

The Image Edge Enhancement core provides three licensing options. After installing the required Xilinx ISE software and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess core functionality with either the example design provided with the Image Edge Enhancement core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the Image Edge Enhancement core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Full implementation support including place-and-route and bitstream generation
- Full functionality in the programmed device with no time outs

Obtaining Your License Key

This section contains information about obtaining a simulation, full system hardware, and full license keys.

Simulation License

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license, do the following:

1. Navigate to the product page for this core.
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

Obtaining a Full License

To obtain a Full license key, you must purchase a license for the core. After doing so, click the “Access Core” link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Performance

Maximum Frequencies

The following are typical clock frequencies for the target devices. The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the Field Programmable Gate Array (FPGA) device, using a different version of Xilinx tools, and other factors.

- Virtex®-7 FPGA: 250 MHz
- Kintex™-7 FPGA: 250 MHz
- Virtex-6 FPGA: 250 MHz
- Spartan®-6 FPGA: 150 MHz

Latency

The propagation delay of the Image Edge Enhancement core is one full scan line and 19 video clock cycles.

Throughput

The Image Edge Enhancement core outputs one YCbCr 4:4:4 sample per clock cycle.

Resource Utilization

Information presented in Tables 1-1 through 1-4 is a guideline to the resource utilization of the Image Edge Enhancement core for Spartan-6, Virtex-6, Virtex-7, and Kintex-7 FPGA families. The General Purpose Processor interface was selected. This core does not use any dedicated IO or clock resources. The design was tested using Xilinx ISE® v11.4i tools with area constraints (see table footnotes) and default tool options.

For an accurate measure of the usage of device resources (for example, block RAMs, flip-flops, and LUTs) for a particular instance, click **View Resource Utilization** in the CORE Generator GUI after generating the core.

Table 1-1: **Resource Utilization and Target Speed for Spartan-6 (1) - xc6slx9,csg324,C,-2**

Data Width	Maximum Number of Columns and Rows	LUT6-FF pairs	LUTs	FFs	RAM36/18	DSP48E1	Clock Frequency (MHz)
8	1024	1,691	1,500	1,268	3 / 0	1	249
8	2200	1,795	1,536	1,307	9 / 0	1	219
10	1024	1,395	1,111	1,419	7 / 1	2	151
10	2200	1,549	1,129	1,459	14 / 1	2	151
12	1024	2,162	1,780	1,644	21 / 0	2	203
12	2200	1,889	1,406	1,628	30 / 0	2	196

1. Speedfile: PRODUCTION 1.20 2011-09-26

Table 1-2: **Resource Utilization and Target Speed for Virtex-6 (1) - xc6vcx75t,ff484,C,-1**

Data Width	Maximum Number of Columns and Rows	LUT6-FF pairs	LUTs	FFs	RAM36/18	DSP48E1	Clock Frequency (MHz)
8	1024	1,237	949	1,185	1 / 1	1	274
8	2200	1,251	1,019	1,226	4 / 1	1	274
10	1024	1,389	1,054	1,363	4 / 0	1	274
10	2200	1,386	1,109	1,404	7 / 0	1	274
12	1024	1,546	1,167	1,541	11 / 0	1	274
12	2200	1,654	1,209	1,582	15 / 0	1	274

1. Speedfile: PRODUCTION 1.10 2011-09-26

Table 1-3: **Resource Utilization and Target Speed for Virtex-7 (1) - xc7k70t,fbg676,C,-1**

Data Width	Maximum Number of Columns and Rows	LUT6-FF pairs	LUTs	FFs	RAM36/18	DSP48E1	Clock Frequency (MHz)
8	1024	1,235	951	1,185	1 / 1	1	370
8	2200	1,281	1,008	1,226	4 / 1	1	352
10	1024	1,322	1,077	1,363	4 / 0	1	344
10	2200	1,523	1,116	1,404	7 / 0	1	335
12	1024	1,565	1,164	1,541	11 / 0	1	309
12	2200	1,598	1,235	1,582	15 / 0	1	335

1. Speedfile: ADVANCED 1.01 2011-09-26

Table 1-4: **Resource Utilization and Target Speed for Kintex-7 (1) - xc7k70t,fbg676,C,-1**

Data Width	Maximum Number of Columns and Rows	LUT6-FF pairs	LUTs	FFs	RAM36/18	DSP48E1	Clock Frequency (MHz)
8	1024	1,318	891	1,185	1 / 1	1	303
8	2200	1,340	1,012	1,226	4 / 1	1	310
10	1024	1,422	1,048	1,363	4 / 0	1	303
10	2200	1,458	1,125	1,404	7 / 0	1	310
12	1024	1,555	1,172	1,541	11 / 0	1	336
12	2200	1,579	1,253	1,582	15 / 0	1	336

1. Speedfile: ADVANCED 1.02 2011-09-26

Chapter 2

Core Interfaces and Register Space

Core Symbol and Port Descriptions

The Image Edge Enhancement core can be configured with three different interface options, each resulting in a slightly different set of ports. The Streaming Video Interface is a set of signals that is common to all three interface options and to all video iPipe cores. It is described in Table 2-1.

Xilinx Streaming Video Interface

The Xilinx Streaming Video Interface (XSVI) is a set of signals common to all of the Xilinx video cores used to stream video data between IP cores. XSVI is also defined as an Embedded Development Kit (EDK) bus type so that the tool can automatically create input and output connections to the core. This definition is embedded in the pCore interface provided with the IP, and it allows an easy way to cascade connections of Xilinx Video Cores. The Image Edge Enhancement IP core uses the following subset of the XSVI signals:.

The Image Edge Enhancement IP Core uses the following sub-set of the XSVI signals:

- video_data
- vblank
- hblank
- active_video

Other XSVI signals on the XSVI input bus, such as video_clk, vsync, hsync, field_id, and active_chr do not affect the function of this core.

Note: These signals are neither propagated, nor driven on the XSVI output of this core.

The following is an example EDK Microprocessor Peripheral Definition (.MPD) file definition:

Input Side:

```
BUS_INTERFACE BUS = XSVI_ENHANCE_IN, BUS_TYPE = TARGET, BUS_STD = XSVI
PORT hblank_i =hblank, DIR=I, BUS=XSVI_ENHANCE_IN
PORT vblank_i =vblank, DIR=I, BUS=XSVI_ENHANCE_IN
PORT active_video_i =active_video,DIR=I, BUS=XSVI_ENHANCE_IN
PORT video_data_i =video_data, DIR=I,VEC=[C_DATA_WIDTH-1:0],
BUS=XSVI_ENHANCE_IN
```

Output Side:

```
BUS_INTERFACE BUS = XSVI_ENHANCE_OUT, BUS_TYPE = INITIATOR, BUS_STD =
XSVI
PORT hblank_o =hblank, DIR=I, BUS=XSVI_ENHANCE_OUT
PORT vblank_o =vblank, DIR=I, BUS=XSVI_ENHANCE_OUT
```

```
PORT active_video_o =active_video,DIR=I, BUS=XSVI_ENHANCE_OUT
PORT video_data_o =video_data,
DIR=I,VEC=[3*C_DATA_WIDTH1:0],BUS=XSVI_ENHANCE_OUT
```

The Image Edge Enhancement IP core is fully synchronous to the core clock, clk. Consequently, the input XSVI bus is expected to be synchronous to the input clock, clk. Similarly, to avoid clock resampling issues, the output XSVI bus for this IP is synchronous to the core clock, clk. The video_clk signals of the input and output XSVI buses are not used.

Constant Interface

The Constant Interface has no ports other than the Streaming Video Interface, as this interface does not provide additional programmability. The Constant Interface Core Symbol is shown in Figure 2-1 and described in Table 2-1. The Streaming Video Interface is a set of signals that is common in all interface options.

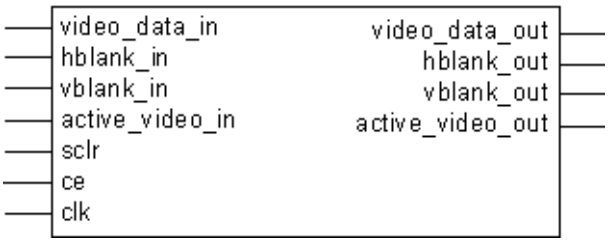

Figure 2-1: **Core Symbol for Constant Interface**

Table 2-1 contains the Constant Interface port descriptions. Detailed descriptions of the ports are provided following the table.

Table 2-1: **Port Descriptions for the Constant Interface**

Port Name	Port Width	Direction	Description
video_data_in	3*WIDTH	IN	Data input bus
hblank_in	1	IN	Horizontal blanking input
vblank_in	1	IN	Vertical blanking input
active_video_in	1	IN	Active video signal input
video_data_out	3*WIDTH	OUT	Data output bus
hblank_out	1	OUT	Horizontal blanking output
vblank_out	1	OUT	Vertical blanking output
active_video_out	1	OUT	Active video signal output
clk	1	IN	Rising-edge clock
ce	1	IN	Clock enable (active high)
sclr	1	IN	Synchronous clear – reset (active high)

- **video_data_in:** This bus contains the luminance and chrominance inputs in the following order from MSB to LSB [Cb ; Cr : Y]. Each component is expected in `WIDTH` bits wide unsigned integer representation.

Bits	*3WIDTH-1:2WIDTH*	*2WIDTH-1:WIDTH*	*WIDTH-1:0*
Video Data Signals	Cb or U	Cr or V	Y

- **hblank_in:** The `hblank_in` signal conveys information about the blank / non-blank regions of video scan lines.
- **vblank_in:** The `vblank_in` signal conveys information about the blank / non-blank regions of video frames, and is used by the Image Edge Enhancement core to detect the end of a frame, when user registers can be copied to active registers to avoid visual tearing of the image.
- **active_video_in:** The `active_video_in` signal is high when valid data is presented at the input. Input data to the core, `video_data_in`, is ignored when `active_video_in` is low.
- **clk - clock:** Master clock in the design, synchronous with, or identical to, the video clock. For the EDK pCore Interface, this port is named `sysgen_clk`.
- **ce - clock enable:** Pulling `CE` low suspends all operations within the core. Outputs are held, and no input signals are sampled except for reset (`SCLR` takes precedence over `CE`). For the EDK pCore Interface, this port is named `sysgen_ce`.
- **sclr - synchronous clear:** Pulling `SCLR` high results in resetting all output pins to zero or their default values. Internal registers within the XtremeDSP™ slice and D-flip-flops are cleared.
- **video_data_out:** This bus contains the luminance and chrominance outputs in the following order from MSB to LSB [Cb ; Cr : Y]. Each component is expected in `WIDTH` bits wide unsigned integer representation.

Bits	*3WIDTH-1:2WIDTH*	*2WIDTH-1:WIDTH*	*WIDTH-1:0*
Video Data Signals	Cb or U	Cr or V	Y

- **hblank_out** and **vblank_out:** The corresponding input signals are delayed so blanking outputs are in phase with the video data output, maintaining the integrity of the video stream.
- **active_video_out:** The `active_video_out` signal is high when valid data is present at the output. When `active_video_out` is low, `video_data_out` is not valid even if it is non-zero.

EDK pCore Interface

The EDK pCore Interface generates AXI4-Lite interface ports in addition to the Streaming Video Signals. The AXI4-Lite signals are automatically connected when the generated pCore is inserted into an EDK project. The Core Symbol for the EDK pCore Interface is shown in Figure 2-2. The Streaming Video Interface is described in the previous section (Table 2-1). For more information on the AXI4-Lite signals, see AXI Reference Guide.

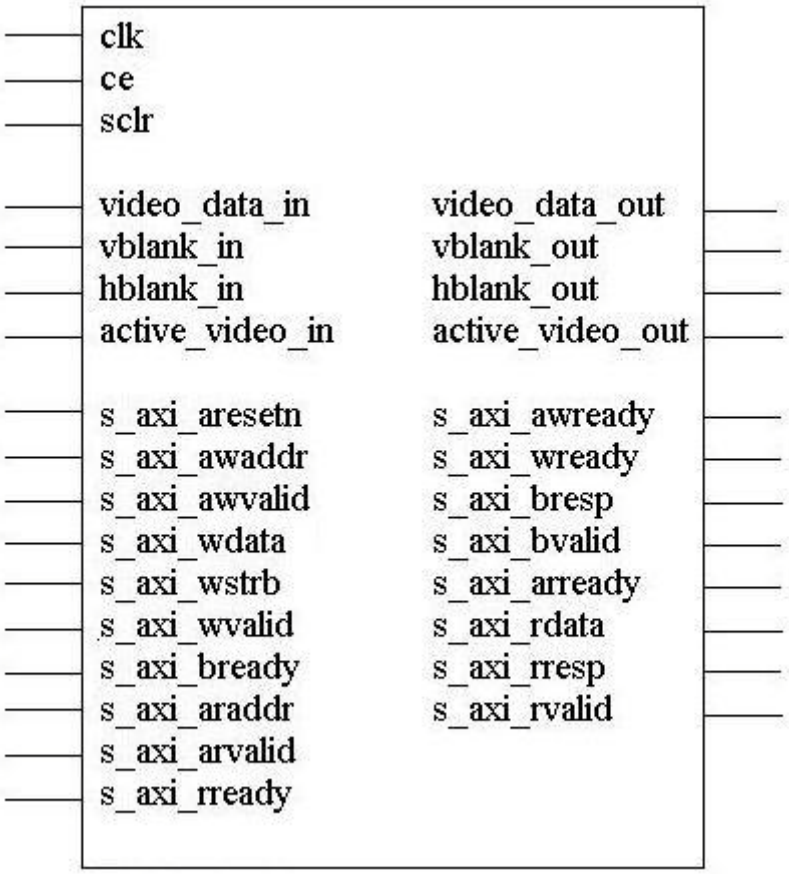

Figure 2-2: **Core Symbol for the EDK pCore Interface**

Many imaging applications include an embedded processor to dynamically control the parameters within an integrated system. CORE Generator software can generate the core with a pCore interface, which can easily be added to an EDK project as a hardware peripheral. This pCore provides a memory- mapped interface for the programmable registers within the core, which are described in Table 2-2.

Table 2-2: **EDK pCore Interface Register Descriptions**

Address Offset (hex)	Register Name	Access Type	Default Value (hex)	Description	
BASEADDR + 0x800	enhance_reg00_control	R/W	0x00000001	Bit 0	Software enable • 0 – Not enabled • 1 – Enabled
				Bit 1	Host processor write done semaphore • 0 – Host processor actively updating registers • 1 – Register update completed by host processor
BASEADDR + 0x804	enhance_reg01_reset	R/W	0x00000000	Bit 0	Software reset • 0 – Not reset • 1 – Reset

Table 2-2: **EDK pCore Interface Register Descriptions *(Cont'd)***

Address Offset (hex)	Register Name	Access Type	Default Value (hex)	Description	
BASEADDR + 0x808	enhance_reg02_gain_H	R/W	From GUI	Gain of Horizontal Sobel filter	Allowed values are 0 to 2 in increments of 0.25 represented by four unsigned bits with two integer bits and two fractional bits Bits: Gain value 0000: 0.00 0001: 0.25 0010: 0.50 0011: 0.75 0100: 1.00 0101: 1.25 0110: 1.50 0111: 1.75 1XXX: 2.00
BASEADDR + 0x80C	enhance_reg03_gain_V	R/W	From GUI	Gain of Vertical Sobel filter	
BASEADDR + 0x810	enhance_reg04_gain_D	R/W	From GUI	Gain of Diagonal Sobel filters	
BASEADDR + 0x814	enhance_reg05_gain_Lap	R/W	From GUI	Gain of Laplacian filter	

All of the registers are readable, enabling the MicroBlaze™ processor to verify writes or read current values contained within the registers. The default values of some of the registers are defined in the Graphical User Interface (GUI) section.

This core supports an enable / disable function. When disabled, the normal operation of the hardware is halted and video signals are not propagated. This function is controlled by setting Software Enable, bit 0 of `enhance_reg00_control` register, to 0. The default value of Software Enable is 1 (enabled).

The in-system reset of the core is controlled by asserting `enhance_reg01_reset` (bit 0), which returns the gains to their default values, specified through the Graphical User Interface when the core is instantiated. The core control signals and output are forced to 0 until the software reset bit is deasserted.

The gain registers are double buffered in hardware to ensure no image tearing happens if the gain values are modified in the active area of a frame. This double buffering provides system control that is more flexible and easier to use by decoupling the register updates from the blanking period, allowing software a much larger window in which to update the parameter values. The updated values for the gain registers are latched into the shadow registers immediately after writing, while the actual gains used are stored in the working registers. The rising edge of `vblank_in` triggers the values from the shadow registers to be copied to the working registers when bit 1 of `enhance_reg00_control` is set to 1. This semaphore bit helps to prevent changing the gains mid-frame.

Any reads of registers return the values stored in the shadow registers.

General Purpose Processor Interface

The General Purpose Processor Interface exposes the gains as a port. The Core Symbol for the General Purpose Processor Interface is shown in Figure 2-3. The Streaming Video Interface is described in the previous section (Table 2-1). The ports are described in Table 2-3.

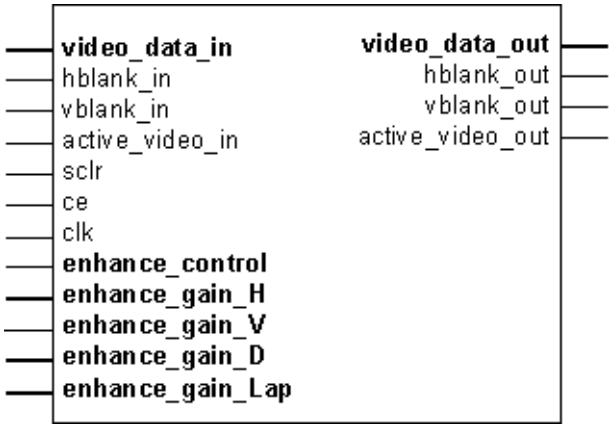

Figure 2-3: **Core Symbol for the General Purpose Processor Interface**

Table 2-3: **Optional Ports for the General Purpose Processor Interface**

Port Name	Port Width	Direction	Description
enhance_control	*2*	IN	Bit 0: Software enable Bit 1: Host processor write done semaphore • 0 – Host processor actively updating registers • 1 – Register update completed by host processor
enhance_gain_H	*4*	IN	Gain of Horizontal Sobel filter Possible bit values: 0000 to 1000 in increments of 0001 Gain is represented as four unsigned bits with two integer bits and two fractional bits (1.0 is represented as 0100)
enhance_gain_V	*4*	IN	Gain of Vertical Sobel filter Possible bit values: 0000 to 1000 in increments of 0001 Gain is represented as four unsigned bits with two integer bits and two fractional bits (1.0 is represented as 0100)
enhance_gain_D	*4*	IN	Gain of Diagonal Sobel filters Possible bit values: 0000 to 1000 in increments of 0001 Gain is represented as four unsigned bits with two integer bits and two fractional bits (1.0 is represented as 0100)
enhance_gain_Lap	*4*	IN	Gain of Laplacian filter Possible bit values: 0000 to 1000 in increments of 0001 Gain is represented as four unsigned bits with two integer bits and two fractional bits (1.0 is represented as 0100)

The General Purpose Processor Interface exposes the gain registers as ports. The General Purpose Processor Interface is provided as an option to design a system with a user-defined bus interface (decoding logic and register banks) to an arbitrary processor.

The gain ports have the double-buffer control mechanism described in the previous section to prevent tearing. However, an external register bank (shadow register bank) has to be supplied by the user-defined bus interface. Values from this register bank (external to the Image Edge Enhancement core) are copied over to the internal registers at the rising edge of `vblank_in` when bit 1 of the `enhance_reg00_control` register is set to '1'.

See General Purpose Processor Interface for more information.

Constant Interface

The Constant Interface does not provide an option for the gains to be changed in system. Also, there is no processor interface and the core is not programmable, but can be reset and enabled / disabled using the `SCLR` and `CE` ports. The ports for the Constant Interface are described in detail in Constant Interface.

Customizing and Generating the Core

This chapter includes information on using Xilinx tools to customize and generate the core.

Graphical User Interface (GUI)

The Image Edge Enhancement core is easily configured to meet user-specific needs through the CORE Generator graphical user interface (GUI). This section provides a quick reference to the parameters that can be configured at generation time. Figure 3-1 shows the main Image Edge Enhancement screen.

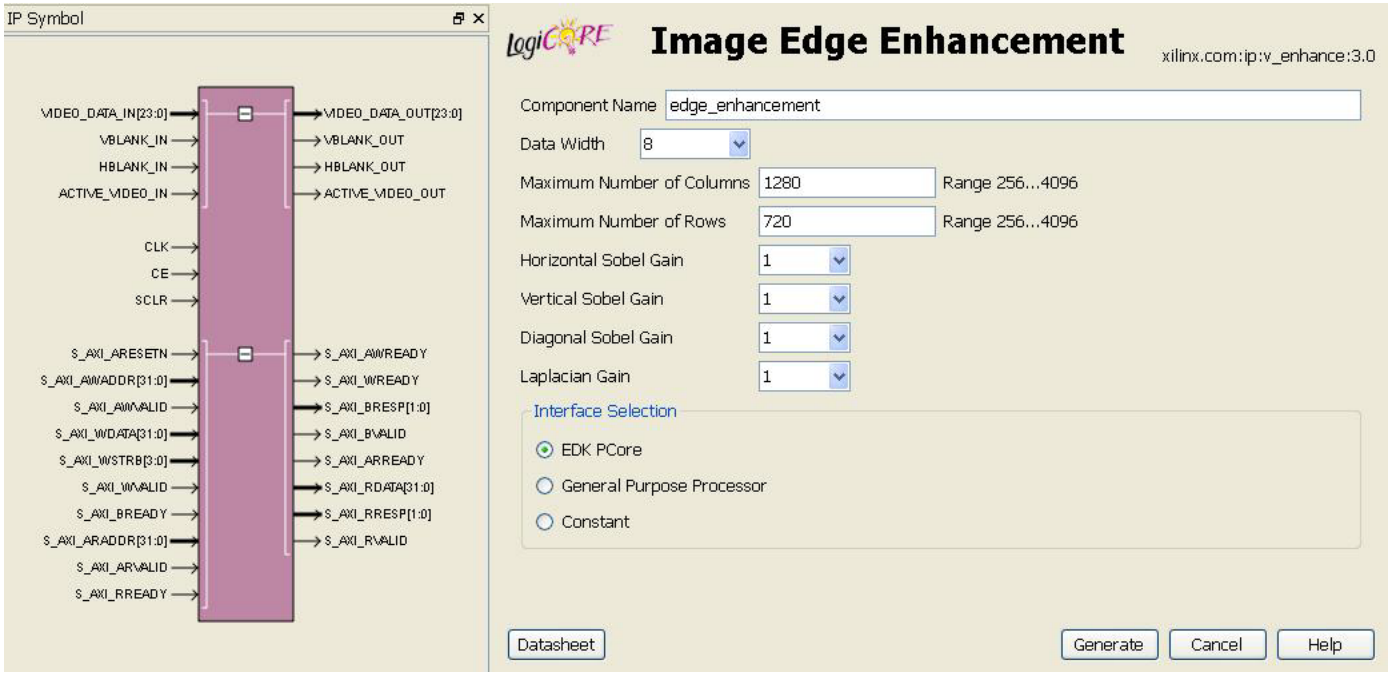

Figure 3-1: **Image Edge Enhancement Main Screen**

The GUI displays a representation of the IP symbol on the left side, and the parameter assignments on the right side, which are described as follows:

- **Component Name:** The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_".
- **Data Width (WIDTH):** Specifies the bit width of the input channel for each component. The allowed values are 8, 10, and 12.

- **Maximum Number of Columns (MAX_COLS)**: Specifies the maximum number of columns that can be processed by the core. Permitted values are from 256 to 4096. Specifying this value is necessary to establish the internal widths of counters and control-logic components as well as the depth of line buffers. Using a tight upper-bound on possible values of MAX_COLS results in optimal block RAM usage. However, feeding the configured Image Edge Enhancement instance timing signals that violate the MAX_COLS constraint leads to data and output timing signal corruption.
- **Maximum Number of Rows (MAX_ROWS):** Specifies the maximum number of rows that can be processed by the core. Permitted values are from 256 to 4096. Specifying this value is necessary to establish the internal widths of counters and control-logic components. Feeding the configured Image Edge Enhancement instance timing signals that violate the MAX_ROWS constraint leads to data and output timing signal corruption.
- **Horizontal Sobel, Vertical Sobel, Diagonal Sobel, and Laplacian Gains**: Specifies the default gain to be applied for each filter. The possible values are 0.0 to 2.0 in increments of 0.25.
- **Interface Selection:** As described in the previous sections, this option allows for the configuration of two different interfaces for the core.
  - **EDK pCore Interface:** CORE Generator software generates a pCore that can be easily imported into an EDK project as a hardware peripheral, and gains can be programmed via a register. Double buffering is used to eliminate tearing of output images. See EDK pCore Interface in Chapter 2.
  - **General Purpose Processor Interface:** CORE Generator software generates a set of ports to be used to program the core. See General Purpose Processor Interface in Chapter 2

Constant Interface: The gains are constant, and therefore no programming is necessary. The constant value is set in the GUI.

Parameter Values in the XCO File

Table 1 defines valid entries for the XCO parameters. Xilinx strongly suggests that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator software GUI to configure the core and perform range and parameter value checking. The XCO parameters are helpful in defining the interface to other Xilinx tools.

XCO Parameter	Default	Valid Values
component_name	edge_enhancement	ASCII text using characters: a..z, 0..9 and "_" starting with a letter. Note: "v_enhance_v3_0" is not allowed.
interface_selection	EDK_Pcore	EDK_Pcore, General_Purpose Processor, Constant
data_width	8	8, 10, 12
diagonal_sobel_gain	1	0 to 2 in 0.25 increments
horizontal_sobel_gain	1	0 to 2 in 0.25 increments
vertical_sobel_gain	1	0 to 2 in 0.25 increments
laplacian_gain	1	0 to 2 in 0.25 increments

XCO Parameter	Default	Valid Values
maximum_number_of_columns	1280	32 to 4095
maximum_number_of_rows	720	32 to 4095

Output Generation

The output files generated from the Xilinx CORE Generator software for the core depend upon whether the interface selection is set to EDK pCore or General Purpose Processor / Constant. The output files are placed in the project directory.

<Component_Name>

- drivers
  - enhance_v3_00_a
    - data
    - build
    - example
    - src
- pcores
  - axi_enhance_v3_00_a
    - data
    - hdl
    - vhdl

File Details

<project directory>

This is the top-level directory. It contains xco and other assorted files.

Name	Description
<component_name>.xco	Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_flist.txt	A text file listing all of the output files produced when the customized core was generated in theCORE Generator software.

<project directory>/<component_name>/pcores/axi_enhance_v3_00_a/data

This directory contains files that EDK uses to define the interface to the pCore.

< project directory>/<component_name>/pcores/axi_enhance_v3_00_a/hdl/vhdl

This directory contains the HDL files that implement the pCore.

< project directory>/<component_name>/drivers/enhance_v3_00_a/data

This directory contains files that SDK uses to define the operation of the pCore's software driver.

< project directory>/<component_name>/drivers/enhance_v3_00_a/example

This directory contains some example code using the pCore's software driver.

< project directory>/<component_name>/drivers/enhance_v3_00_a/src

This directory contains the source code of the pCore's software driver.

Name	Description
enhance.c	Provides the API access to all features of the device driver.
enhance.h	Provides the API access to all features of the device driver.

General Purpose Processor or Constant Interface Files

When the interface selection is set to General Purpose Processor, CORE Generator will output the core as a netlist that can be inserted into a processor interface wrapper or instantiated directly in an HDL design. The output is placed in the <project director>.

File Details

The CORE Generator output consists of some or all the following files.

Name	Description
<component_name>_readme.txt	Readme file for the core.
<component_name>.ngc	The netlist for the core.
<component_name>.veo	The HDL template for instantiating the core.
<component_name>.vho	
<component_name>.v	The structural simulation model for the core. It is used for functionally simulating the core.
<component_name>.vhd	
<component_name>.xco	Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.

Designing with the Core

The Sobel operators are defined in Equations 4-1, 4-2, and 4-3.

$$\text{Horizontal Sobel} = \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{Equation 4-1}$$

$$\text{Vertical Sobel} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{Equation 4-2}$$

$$\text{Diagonal Sobels} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \quad \text{Equation 4-3}$$

The Laplacian is defined in Equation 4-4.

$$\text{Laplacian} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{Equation 4-4}$$

Human visual systems detect the boundary of objects best when they are accompanied by sudden changes in brightness. The edge enhancement core exploits this and enhances only the luminance channel. This has the added benefit of eliminating color shifts at the boundary of objects, which are common when enhancing the chrominance components by similar methods. The luminance component is processed through the core in two dimensions using two line buffers. The chrominance components are passed through the core with the proper delay to match luminance processing. This core can accept chrominance components represented as signed or unsigned integers with or without the 128 offset.

Defining Gains

The amount and direction of the edge enhancement can be controlled through programmable gains `gH` (horizontal), `gV` (vertical), `gD` (diagonal), and `gLap` (Laplacian). Here, a vertical edge is defined as a feature running from top to bottom of an image. Similarly, a horizontal edge runs from left to right across the image. The diagonal direction covers both upper left to lower right and upper right to lower left diagonals.

Gains can be set to values in the range of 0.0 to 2.0 (see the EDK pCore Interface and General Purpose Processor Interface sections for details). If a particular direction is not desired, that gain can be set to zero to eliminate emphasis in that direction. For example, if vertical edges do not need to be enhanced, the gain `gV` should be set to zero.

Additionally, there is an image content dependent gain, `K`, used to modify the Sobel and Laplacian output. In areas of the image that are smooth and of low contrast, the gain is low to avoid emphasizing noise. This gain is automatically and dynamically calculated by the core on a pixel basis, and it is designed to produce a good compromise between enhancement of features and undesired noise.

If the total gain used [(gH+gV+gD+gLap)*K] exceeds 1.0, clipping and clamping circuitry limits the enhancement of the edge content. Setting the gains with values greater than 1.0 allows over-enhancing the image to produce special effects like embossing.

Over-emphasis of edges may bring out noise at the edge transitions, and therefore this core may be used in conjunction with noise reduction cores such as the Image Noise Reduction LogiCORE IP to improve the results.

General Design Guidelines

Control Signals and Timing

Figure 4-1 shows a typical timing example with two frames of data.

The propagation delay of the Image Edge Enhancement core is one full scan line and 19 video clock cycles. The output timing signals (`vblank_out`, `hblank_out`, and `active_video_out`) are delayed appropriately so that the output video data is framed correctly by the timing signals.

Deasserting `CE` suspends processing, which may be useful for data-throttling, to temporarily cease processing of a video stream to match the delay of other processing components.

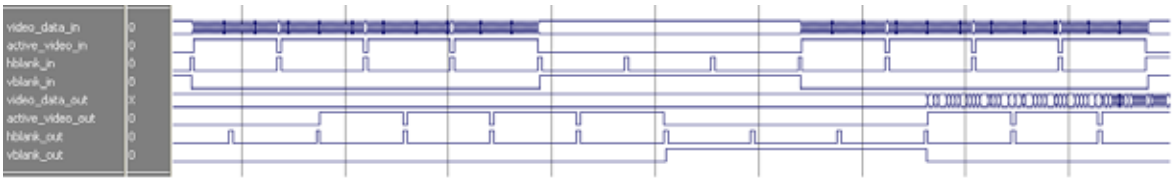

Figure 4-1: **Timing Example**

The control signals `vblank_out`, `hblank_out`, and `active_video_out` are created using a timing detector and generator within the core. The internal timing module assumes the following:

- One horizontal blanking period per row
- One vertical blanking period per frame
- A minimum active frame size of four rows and eight columns
- A minimum horizontal blanking period of two columns
- A minimum vertical blanking period of three rows

During the detection of the timing control signals, the core cannot guarantee the correct video data output. Therefore, the data output, `video_data_out`, of the first frame of data is set to zero even though `active_video_out` is high.

When SCLR is asserted, all data and control signal outputs are forced to zero. If the input control signal was high at the time SCLR was asserted, the corresponding output control signal goes low and stays low until the next expected rising edge.

Clocking

The Image Edge Enhancement core has one clock ("clk") that is used to clock the entire core. This includes the AXI interface and the core logic.

Resets

The Image Edge Enhancement core has one reset ("sclr") that is used for the entire core. The reset is active high.

Protocol Description

For the pCore version of the Image Edge Enhancement core, the register interface is compliant with the AXI4-Lite interface.

Constraining the Core

Required Constraints

The clk pin should be constrained at the maximum pixel clock rate desired for the video stream.

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for this core. This core has not been characterized for use in low power devices.

Clock Frequencies

The clk pin should be run at the required pixel clock frequency for the Edge Enhancement core. See Maximum Frequency in Performance in Chapter 1.

Clock Management

There is only one clock for this core. For pCore users, the AXI interconnect handles the cross clock domain crossing.

Clock Placement

There are no specific clock placement requirements for this core.

Banking

There are no specific banking rules for this core.

Transceiver Placement

There are no transceivers used in this core.

I/O Standard and Placement

There are no specific I/O standard or placement requirements.

XILINX®

Chapter 6

Detailed Example Design

Overview

This describes how to use the files that come with this demo testbench package for Image Edge Enhancement v3.0.

This demo testbench is provided as a simple introductory package that enables core users to observe the core generated by Coregen operating in a waveform simulator. The user is encouraged to observe core-specific aspects in the waveform, make simple modifications to the test conditions, and observe the changes in the waveform.

For more information about Image Edge Enhancement v3.0, refer to the following link: http://www.xilinx.com/products/ipcenter/EF-DI-IMG-ENHANCE.htm

Design File Hierarchy

The directory structure underneath this top-level folder is described below:

- Expected

  Contains the pre-generated expected/golden data used by the testbench to compare actual output data.

- Stimuli

  Contains the pre-generated input data used by the testbench to stimulate the core (including register programming values).

- Results

  Actual output data will be written to a file in this folder.

- src

  Contains the .vhd & .xco files of the core.

- The .vhd file is a netlist generated using Coregen.

  You can regenerate a new netlist using the .xco file in Coregen.

- tb_src

  Contains the top-level testbench design.

  This directory also contains other packages used by the testbench.

- isim_wave.wcfg - Waveform configuration for ISIM
- mti_wave.do - Waveform configuration for ModelSim
- run_isim.bat - Runscript for iSim in Windows OS
- run_isim.sh - Runscript for iSim in Linux OS

- run_mti.bat - Runscript for ModelSim in Windows OS
- run_mti.sh - Runscript for ModelSim in Linux OS

Operating Instructions

Simulation using ModelSim for Linux :

- From the console, Type "source run_mti.sh".

Simulation using ModelSim for Windows :

- Double click on "run_mti.bat" file.

Simulation using iSim for Linux :

- From the console, Type "source run_isim.sh".

Simulation using iSim for Windows :

- Double click on "run_isim.bat" file.

Support

To obtain technical support for this reference design, go to www.xilinx.com/support to locate answers to known issues in the Xilinx Answers Database or to create a WebCase.

XILINX®

Appendix A

Verification, Compliance, and Interoperability

Simulation

A highly parameterizable test bench used to test the Image Edge Enhancement core. Testing includes the following:

- Register accesses
- Testing the vertical, horizontal, and diagonal Sobel filters
- Testing the Laplacian filter
- Testing of all the filter gains
- Testing of various frame sizes
- Testing of various data widths

Hardware Testing

The Image Edge Enhancement core has been tested in a variety of hardware platforms at Xilinx to represent a variety of parameterizations.

A test design was developed for the core that incorporated a MicroBlaze processor, AXI4-LITE Interface and various other peripherals.

XILINX®

Appendix B

Migrating

Special Considerations when Migrating to AXI

The Image Edge Enhancement v3.0 interface changed from the PLB processor interface to the EDK pCore AXI4-Lite interface. As a result, all of the PLB-related connections have been replaced with an AXI4-Lite interface. For more information, see the AXI Reference Guide.

Appendix C

Debugging

Evaluation Core Timeout

The Image Edge Enhancement hardware evaluation core times out after approximately 8 hours of operation. The output is driven to zero. This results in a dark-green screen for YUV color systems.

See Solution Centers in Appendix F for information helpful to the debugging progress.

Application Software Development

Figure D-1 shows a software flow diagram for updating registers during the operation of the core. See the EDK pCore Interface.

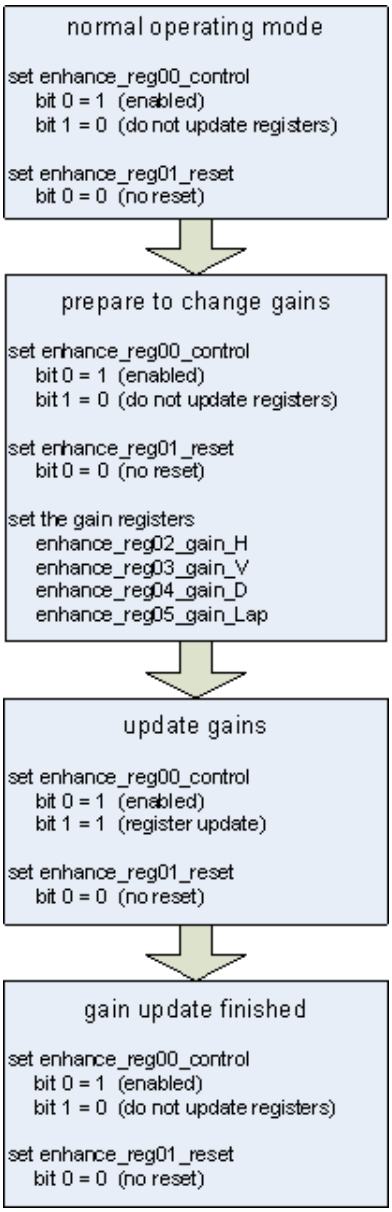

Figure D-1: **Image Edge Enhancement Programming Flow Chart**

Programmer's Guide

The software API is provided to allow easy access to the Image Edge Enhancement pCore's registers defined in Table 2-1. To utilize the API functions provided, the following two header files must be included in the user C code:

```
#include "enhance.h"
#include "xparameters.h"
```

The hardware settings of your system, including the base address of your Image Edge Enhancement core, are defined in the `xparameters.h` file. The `enhance.h` file contains the macro function definitions for controlling the Image Edge Enhancement pCore.

For examples on API function calls and integration into a user application, the drivers subdirectory of the pCore contains a file, `example.c`, in the `enhance_v3_00_a/example` subfolder. This file is a sample C program that demonstrates how to use the Image Edge Enhancement pCore API.

EDK pCore API Functions

This section describes the functions included in the C driver (enhance.c and enhance.h) generated for the EDK pCore API.

ENHANCE_Enable(uint32 BaseAddress);

- This macro enables an Image Edge Enhancement instance.
- BaseAddress is the Xilinx EDK base address of the Image Edge Enhancement core (from xparameters.h).

ENHANCE_Disable(uint32 BaseAddress);

- This macro disables an Image Edge Enhancement instance.
- BaseAddress is the Xilinx EDK base address of the Image Edge Enhancement core (from xparameters.h).

ENHANCE_Reset(uint32 BaseAddress);

- This macro resets an Image Edge Enhancement instance. This reset effects the core immediately, and may cause image tearing. Reset affects the gain registers, forces video_data_out to 0, and forces timing signal outputs to their reset state until ENHANCE_ClearReset() is called.
- BaseAddress is the Xilinx EDK base address of the Image Edge Enhancement core (from xparameters.h)

ENHANCE_ClearReset(uint32 BaseAddress);

- This macro clears the reset flag of the core, which allows it to re-sync with the input video stream and return to normal operation.
- BaseAddress is the Xilinx EDK base address of the Image Edge Enhancement core (from xparameters.h).

Reading and Writing pCore Registers

Each software register defined in Table 1 has a constant defined in enhance.h that is set to the offset for that register. Reading a value from a register uses the base address and offset for the register:

```
Xuint32 value = ENHANCE_ReadReg(XPAR_ENHANCE_0_BASEADDR, ENHANCE_REG03_GAIN_H);
```

This macro returns the 32-bit unsigned integer value of the register. The definition of this macro is

ENHANCE_ReadReg(uint32 BaseAddress, uint32 RegOffset)

- Read the given register.
- BaseAddress is the Xilinx EDK base address of the Image Edge Enhancement core (from xparameters.h).
- RegOffset is the register offset of the register (defined in Table 1).

To write to a register, use the ENHANCE_WriteReg() function using the base address of the Image Edge Enhancement pCore instance (from xparameters.h), the offset of the desired register, and the data to write. For example:

```
ENHANCE_WriteReg(XPAR_ENHANCE_0_BASEADDR, ENHANCE_REG03_GAIN_H, 1);
```

The definition of this macro is:

ENHANCE_WriteReg(uint32 BaseAddress, uint32 RegOffset, uint32 Data)

- Write the given register.
- BaseAddress is the Xilinx EDK base address of the Image Edge Enhancement core (from xparameters.h).
- RegOffset is the register offset of the register (defined in Table 1).
- Data is the 32-bit value to write to the register.

ENHANCE_RegUpdateEnable(uint32 BaseAddress);

- Calling RegUpdateEnable causes the Image Edge Enhancement to start using the updated gain values on the next rising edge of VBlank_in. The user must manually disable the register update after a sufficient amount of time to prevent continuous updates.
- This function only works when the Image Edge Enhancement core is enabled.
- BaseAddress is the Xilinx EDK base address of the Image Edge Enhancement core (from xparameters.h)

ENHANCE_RegUpdateDisable(uint32 BaseAddress);

- Disabling the Register Update prevents the Image Edge Enhancement gain registers from updating. Xilinx recommends that the Register Update be disabled while writing to the registers in the core, until the write operation is complete. While disabled, writes to the registers are stored, but do not affect the core's behavior.
- This function only works when the Image Edge Enhancement core is enabled.
- BaseAddress is the Xilinx EDK base address of the Image Edge Enhancement core (from xparameters.h)

XILINX®

Appendix E

C Model Reference

Unpacking and Model Contents

Unzip the `v_enhance_v3_0_bitacc_model.zip` file, containing the bit accurate models for the Image Edge Enhancement IP Core. This creates the directory structure and files in Table E-1.

Table E-1: **Directory Structure and Files of the Image Edge Enhancement v3.0 Bit Accurate C Model**

File Name	Contents
README.txt	Release notes
pg003_v_enhance.pdf	LogiCORE IP Image Edge Enhancement Bit Accurate C Model User Guide
v_enhance_v3_0_bitacc_cmodel.h	Model header file
rgb_utils.h	Header file declaring the RGB image / video container type and support functions
yuv_utils.h	Header file declaring the YUV (.yuv) image file I/O functions
bmp_utils.h	Header file declaring the bitmap (.bmp) image file I/O functions
video_utils.h	Header file declaring the generalized image / video container type, I/O and support functions
run_bitacc_cmodel.c	Example code calling the C model
kodim19_128x192.bmp	128x192 sample test image of the lighthouse image from the True Color Kodak test images
/lin64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms
libIp_v_enhance_v3_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_enhance_v3_0_bitacc_cmodel.so
/lin32	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms
libIp_v_enhance_v3_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_enhance_v3_0_bitacc_cmodel.so
/nt32	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms.

Table E-1: **Directory Structure and Files of the Image Edge Enhancement v3.0 Bit Accurate C Model**

libIp_v_enhance_v3_0_bitacc_cmodel.lib	Precompiled library file for win32 compilation
/nt64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms.
libIp_v_enhance_v3_0_bitacc_cmodel.lib	Precompiled library file for win64 compilation

Installation

For Linux, make sure these files are in a directory that is in your $LD_LIBRARY_PATH environment variable:

- `libIp_v_enhance_v3_0_bitacc_cmodel.so`
- `libstlport.so.5.1`

Software Requirements

The Image Edge Enhancement v3.0 C models were compiled and tested with the software listed in Table E-2.

Table E-2: **Compilation Tools for the Bit Accurate C Models**

Platform	C Compiler
32- and 64-bit Linux	GCC 4.1.1
32- and 64-bit Windows	Microsoft Visual Studio 2005

Using the C Model

The bit accurate C model is accessed through a set of functions and data structures that are declared in the `v_enhance_v3_0_bitacc_cmodel.h` file.

Before using the model, the structures holding the inputs, generics and output of the Image Edge Enhancement instance must be defined:

```
struct  xilinx_ip_v_enhance_v3_0_generics enhance_generics;
struct  xilinx_ip_v_enhance_v3_0_inputs   enhance_inputs;
struct  xilinx_ip_v_enhance_v3_0_outputs  enhance_outputs;
```

The declaration of these structures is in the `v_enhance_v3_0_bitacc_cmodel.h` file.

Table E-3 lists the generic parameters taken by the Image Edge Enhancement v3.0 IP core bit accurate model, as well as the default values. For an actual instance of the core, these parameters can only be set in generation time through the CORE Generator™ GUI.

Table E-3: **Model Generic Parameters and Default Values**

Generic variable	Type	Default Value	Range	Description
DATA_WIDTH	int	8	8,10,12	Data width

Calling `xilinx_ip_v_enhance_v3_0_get_default_generics(&enhance_generics)` initializes the generics structure with the Image Edge Enhancement GUI defaults, listed in Table E-3.

Direction gain can also be set dynamically through the pCore and General Purpose Processor interfaces. Consequently, these values are passed as inputs to the core, along with the actual test image, or video sequence (Table E-4).

Table E-4: **Core Generic Parameters and Default Values**

Input Variable	Type	Default Value	Range	Description
video_in	video_struct	null	N / A	Container to hold input image or video data.[1]
gain_h	int	1	Allowed values are 0 to 2 in increments of 0.25	Horizontal gain
gain_v	int	1	Allowed values are 0 to 2 in increments of 0.25	Vertical gain
gain_d	int	1	Allowed values are 0 to 2 in increments of 0.25	Diagonal gain
gain_lap	int	1	Allowed values are 0 to 2 in increments of 0.25	Laplacian filter gain

[1] For the description of the input structure, see Initializing the Image Edge Enhancement Input Video Structure.

The structure `enhance_inputs` defines the values of run time parameters and the actual input image. Calling `xilinx_ip_v_enhance_v3_0_get_default_inputs(&enhance_generics, &enhance_inputs)` initializes the input structure with the default values (see Table E-4).

Note: The `video_in` variable is not initialized because the initialization depends on the actual test image to be simulated. Chapter 4, C Model Example Code describes the initialization of the `video_in` structure.

After the inputs are defined, the model can be simulated by calling this function:

```
int xilinx_ip_v_enhance_v3_0_bitacc_simulate(
struct xilinx_ip_v_enhance_v3_0_generics* generics,
struct xilinx_ip_v_enhance_v3_0_inputs*   inputs,
struct xilinx_ip_v_enhance_v3_0_outputs*  outputs).
```

Results are included in the outputs structure, which contains only one member, type `video_struct`. After the outputs are evaluated and saved, dynamically allocated memory for input and output video structures must be released by calling this function:

```
void xilinx_ip_v_enhance_v3_0_destroy(
struct xilinx_ip_v_enhance_v3_0_inputs *input,
struct xilinx_ip_v_enhance_v3_0_outputs *output).
```

Successful execution of all provided functions, except for the destroy function, return value 0. A non-zero error code indicates that problems occurred during function calls.

Image Edge Enhancement Input and Output Video Structure

Input images or video streams can be provided to the Image Edge Enhancement v3.0 reference model using the `video_struct` structure, defined in `video_utils.h`:

```
struct video_struct{
  int       frames, rows, cols, bits_per_component, mode;
  uint16*** data[5]; };
```

Table E-5: **Member Variables of the Video Structure**

Member Variable	Designation
frames	Number of video / image frames in the data structure.
rows	Number of rows per frame. Pertaining to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through all frames of the video stream. However different planes, such as y, u and v can have different dimensions.
cols	Number of columns per frame. Pertaining to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through all frames of the video stream. However different planes, such as y, u and v can have different dimensions.
bits_per_component	Number of bits per color channel / component.All image planes are assumed to have the same color / component representation. Maximum number of bits per component is 16.
mode	Contains information about the designation of data planes. Named constants to be assigned to mode are listed in Table E-6.
data	Set of five pointers to three dimensional arrays containing data for image planes. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col].

Table E-6: **Named Video Modes with Corresponding Planes and Representations**

Mode	Planes	Video Representation
FORMAT_MONO	1	Monochrome – Luminance only
FORMAT_RGB	3	RGB image / video data
FORMAT_C444	3	444 YUV, or YCrCb image / video data
FORMAT_C422	3	422 format YUV video, (u, v chrominance channels horizontally sub-sampled)
FORMAT_C420	3	420 format YUV video, (u, v sub-sampled both horizontally and vertically)
FORMAT_MONO_M	3	Monochrome (Luminance) video with Motion
FORMAT_RGBA	4	RGB image / video data with alpha (transparency) channel
FORMAT_C420_M	5	420 YUV video with Motion
FORMAT_C422_M	5	422 YUV video with Motion
FORMAT_C444_M	5	444 YUV video with Motion
FORMAT_RGBM	5	RGB video with Motion

The Image Edge Enhancement core supports the mode FORMAT_C444.

Initializing the Image Edge Enhancement Input Video Structure

The easiest way to assign stimuli values to the input video structure is to initialize it with an image or video. The `bmp_util.h` and `video_util.h` header files packaged with the bit accurate C models contain functions to facilitate file I/O.

YUV Image Files

The header `yuv_utils.h` file declares functions that help access files in standard YUV format. It operates on images with three planes (Y, U and V). The following functions operate on arguments of type `yuv8_video_struct`, which is defined in `yuv_utils.h`.

```
int write_yuv8(FILE *outfile, struct yuv8_video_struct *yuv8_video);
int read_yuv8(FILE *infile, struct yuv8_video_struct *yuv8_video);
```

Exchanging data between `yuv8_video_struct` and general `video_struct` type frames / videos is facilitated by these functions:

```
int copy_yuv8_to_video(struct yuv8_video_struct* yuv8_in,
                       struct video_struct* video_out );
int copy_video_to_yuv8(struct video_struct* video_in,
                       struct yuv8_video_struct* yuv8_out );
```

Note: All image/video manipulation utility functions expect both input and output structures initialized; for example, pointing to a structure that has been allocated in memory, either as static or dynamic variables. Moreover, the input structure must have the dynamically allocated container (data or r, g, b) structures already allocated and initialized with the input frame(s). If the output container structure is pre-allocated at the time of the function call, the utility functions verify and issue an error if the output container size does not match the size of the expected output. If the output container structure is not pre-allocated, the utility functions create the appropriate container to hold results.

Binary Image/Video Files

The `video_utils.h` header file declares functions that help load and save generalized video files in raw, uncompressed format.

```
int read_video( FILE* infile,  struct video_struct* in_video);
int write_video(FILE* outfile, struct video_struct* out_video);
```

These functions serialize the `video_struct` structure. The corresponding file contains a small, plain text header defining, "Mode", "Frames", "Rows", "Columns", and "Bits per Pixel". The plain text header is followed by binary data, 16-bits per component in scan line continuous format. Subsequent frames contain as many component planes as defined by the video mode value selected. Also, the size (rows, columns) of component planes can differ within each frame as defined by the actual video mode selected.

Working with Video_struct Containers

The `video_utils.h` header file defines functions to simplify access to video data in `video_struct`.

```
int video_planes_per_mode(int mode);
int video_rows_per_plane(struct video_struct* video, int plane);
int video_cols_per_plane(struct video_struct* video, int plane);
```

The `video_planes_per_mode` function returns the number of component planes defined by the mode variable, as described in Table E-6. The `video_rows_per_plane` and `video_cols_per_plane` functions return the number of rows and columns in a given plane of the selected video structure. The following example demonstrates using these functions in conjunction to process all pixels within a video stream stored in the `in_video` variable:

```
for (int frame = 0; frame < in_video->frames; frame++) {
  for (int plane = 0; plane < video_planes_per_mode(in_video->mode); plane++) {
    for (int row = 0; row < rows_per_plane(in_video,plane); row++) {
      for (int col = 0; col < cols_per_plane(in_video,plane); col++) {
   // User defined pixel operations on
// in_video->data[plane][frame][row][col]
      }
    }
  }
}
```

C Model Example Code

An example C file, `run_bitacc_cmodel.c`, is provided to demonstrate the steps required to run the model. After following the compilation instructions, run the example executable. The executable takes the path / name of the input file and the path / name of the output file as parameters. If invoked with insufficient parameters, this help message is issued:

```
Usage: run_bitacc_cmodel in_file out_file
in_file     : path/name of the input  (BIN file)
out_file    : path/name of the output (BIN file)
```

The structure of .bin files are described in Binary Image / Video Files.

To ease modifying and debugging the provided top-level demonstrator using the built-in debugging environment of Visual Studio, the top-level command line parameters can be specified through the Project Property Pages using these steps:

1. In the Solution Explorer pane, right-click the project name and select Properties in the context menu.
2. Select Debugging on the left pane of the Property Pages dialog box.
3. Enter the paths and file names of the input and output images in the Command Arguments field.

Compiling Image Edge Enhancement C Model with Example Wrapper

Linux (32-bit and 64-bit)

To compile the example code, perform these steps:

1. Set your `$LD_LIBRARY_PATH` environment variable to include the root directory where you unzipped the model zip file using a command such as:

   ```
   setenv LD_LIBRARY_PATH <unzipped_c_model_dir>:${LD_LIBRARY_PATH}
   ```

2. Copy these files from the /lin64 directory to the root directory:

   `libstlport.so.5.1`

   `libIp_v_enhance_v3_0_bitacc_cmodel.so`

3. In the root directory, compile using the GNU C Compiler with this command:

```
gcc -m32 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
run_bitacc_cmodel -L. -lIp_v_enhance_v3_0_bitacc_cmodel
-Wl,-rpath,.
```

```
gcc –m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
run_bitacc_cmodel -L. -lIp_v_enhance_v3_0_bitacc_cmodel
-Wl,-rpath,.
```

Windows (32-bit and 64-bit)

The precompiled library `v_enhance_v3_0_bitacc_cmodel.lib`, and top-level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. An example procedure is provided here using Microsoft Visual Studio.

1. In Visual Studio, create a new, empty Win32 Console Application project.
2. As existing items, add:
   a. `libIp_v_enhance_v3_0_bitacc_cmodel.lib` to the Resource Files folder of the project
   b. `run_bitacc_cmodel.c` to the Source Files folder of the project
   c. `v_enhance_v3_0_bitacc_cmodel.h` to the Header Files folder of the project
3. After the project is created and populated, it must be compiled and linked (built) to create a win32 executable. To perform the build step, select "Build Solution" from the Build menu. An executable matching the project name has been created either in the Debug or Release subdirectories under the project location based on whether "Debug" or "Release" has been selected in the "Configuration Manager" under the Build menu.

XILINX®

Appendix F

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

http://www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

These documents provide supplemental material useful with this user guide:

1. AXI Reference Guide.

Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the IP Release Notes Guide (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Ordering Information

The Image Enhancement core is provided under the the Xilinx Core License Agreement and can be generated using the Xilinx® CORE Generator™ system. The CORE Generator system is shipped with Xilinx ISE® Design Suite software.

A simulation evaluation license for the core is shipped with the CORE Generator system. To access the full functionality of the core, including FPGA bitstream generation, a full license must be obtained from Xilinx. For more information, visit the product page for this core.

Contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx IP Center.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release.

Notice of Disclaimer