LogiCORE IP Image Noise Reduction v3.0

Product Guide

PG011 October 19, 2011

XILINX®

Table of Contents

LogiCORE IP Image Noise Reduction v3.0

Introduction

The Xilinx LogiCORE™ IP Image Noise Reduction core is an easy-to-use IP block for reducing noise within each frame of video. The core has a programmable, edge-adaptive smoothing function to change the characteristics of the filtering in real-time.

Features

- Support for:
  - High-definition (1080p60) resolutions
  - Up to 4096 total pixels and 4096 total rows
- In-system update of smoothing filters
- Selectable processor interface
  - EDK pCore
  - General Purpose Processor
  - Constant Interface
- Support for 8-bit, 10-bit, or 12-bit input and output precision
- YCrCb or YUV 444 input and output
- Xilinx Streaming Video Interface (XSVI) bus simplifies connecting to other video IP

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

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 Image Noise Reduction product page.
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

Chapter 1

Overview

The Image Noise Reduction Core performs noise reduction by applying a smoothing filter to the image. The smoothing filter is applied depending on the edge content in the image. Near edges, the gain is low so that edge have less smoothing applied.

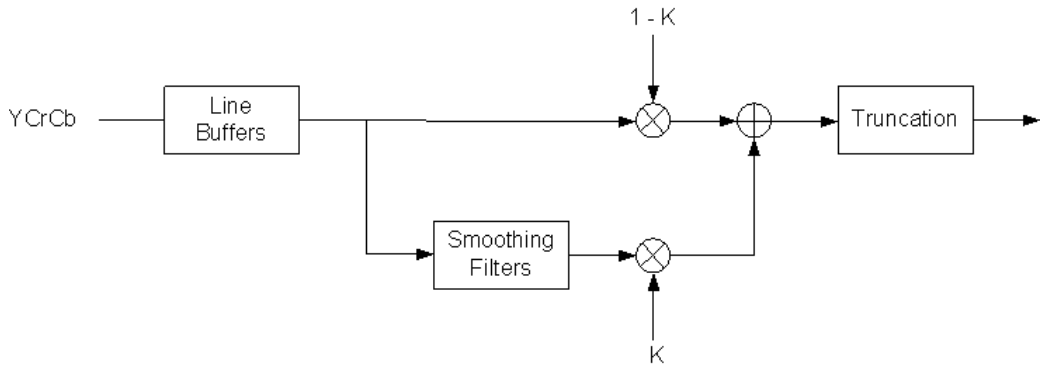

Figure 1-1: **Image Noise Reduction**

Standards Compliance

The Image Noise Reduction core is compliant with the AXI4-Lite interconnect standard as defined in UG761, *Xilinx AXI Reference Guide*.

Feature Summary

The Image Noise Reduction Core provides data from video streams of a maximum resolution of 4096 columns by 4096 rows and supports the bandwidth necessary for high-definition (1080p60) resolutions. The strength of the smoothing is run-time programmable via an EDK pCore interface or a generic General Purpose Processor interface.

Applications

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

Licensing

The Image Noise Reduction core provides the following three licensing options:

- Simulation Only
- Full System Hardware Evaluation
- Full

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 Noise Reduction 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.)

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

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 Noise Reduction core using the example design and demonstration test bench provided with the 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.

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.

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

To obtain a Full license key, you must purchase a license for the core. Click on the “Order” link on the Xilinx.com IP core product page for information on purchasing a license for this core. After doing so, go to the core product page and click the “How do I generate a license key to activate this core?” link 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

The following sections detail the performance characteristics of the Image Noise Reduction core.

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 Noise Reduction core is one full scan line and 18 video clock cycles.

Throughput

The Image Noise Reduction core produces statistics in real time for the data it consumes. If timing constraints are met, the throughput is equal to the rate at which video data is written into the core. In numeric terms, 1080P/60 represents an average data rate of 124.4 Mpixels/sec or a burst data rate of 148.5 Mpixels/sec.

Resource Utilization

Information presented in Table 1-1 through Table 1-4 is a guideline to the resource utilization of the Image Noise Reduction core for Virtex-7, Kintex-7, Virtex-6, and Spartan-6 FPGA families. This core does not use any dedicated I/O or clock resources. The design was tested using Xilinx ISE® software with area constraints 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 interface after generating the core.

Table 1-1: **Virtex-7 Resource Usage**

Data Width	Max Number of Columns and Rows	LUTs	FFs	LUT6-FF PAIR	RAM36	RAM18	DSP48E1	Clock Frequency (MHz)
8	1024	1402	1612	1671	1	1	3	329
8	2200	1484	1653	1839	5	1	3	293
10	1024	1637	1894	2048	2	0	3	335
10	2200	1715	1935	2111	6	2	3	335
12	1024	1900	2188	2302	2	0	3	313
12	2200	1924	2228	2405	8	0	3	313

1. Device, Part, Speed: XC7V585T,-1 (ADVANCED 1.02 2011-09-27)

Table 1-2: **Kintex-7 Resource Usage**

Data Width	Max Number of Columns and Rows	LUTs	FFs	LUT6-FF PAIR	RAM36	RAM18	DSP48E1	Clock Frequency (MHz)
8	1024	1400	1612	1722	1	1	3	295
8	2200	1508	1653	1793	5	1	3	328
10	1024	1610	1894	2077	2	0	3	320
10	2200	1706	1935	2156	6	2	3	304
12	1024	1909	2188	2328	2	0	3	337
12	2200	1947	2228	2380	8	0	3	320

1. Device, Part, Speed: XC7K70T,-1 (ADVANCED 1.02 2011-09-27)

Table 1-3: **Virtex-6 Resource Usage**

Data Width	Max Number of Columns and Rows	LUTs	FFs	LUT6-FF PAIR	RAM36	RAM18	DSP48E1	Clock Frequency (MHz)
8	1024	1415	1612	1744	1	1	3	346
8	2200	1494	1653	1762	5	1	3	307
10	1024	1650	1894	2028	2	0	3	307
10	2200	1737	1935	2063	6	2	3	315
12	1024	1883	2188	2242	2	0	3	322
12	2200	1959	2229	2448	8	0	3	330

1. Device, Part, Speed: XC6VLX75T,-1 (PRODUCTION 1.15 2011-09-27)

Table 1-4: **Spartan-6 Resource Usage**

Data Width	Max Number of Columns and Rows	LUTs	FFs	LUT6-FF PAIR	RAM36	RAM18	DSP48E1	Clock Frequency (MHz)
8	1024	1849	1701	2167	3	0	3	230
8	2200	1874	1733	2103	12	0	3	225
10	1024	2102	1991	2345	3	1	3	236
10	2200	2155	2031	2452	15	0	3	216
12	1024	2417	2417	2802	4	0	3	216
12	2200	2501	2501	2758	8	0	3	225

1. Device, Part, Speed: XC6SLX150,-2 (PRODUCTION 1.20 2011-09-27)

XILINX®

Chapter 2

Core Interfaces and Register Space

This chapter provides detailed descriptions for each interface. In addition, detailed information about configuration and control registers is included.

Port Descriptions

Processor Interfaces

The Image Noise Reduction core supports the following three processor interface options:

- EDK pCore Interface
- General Purpose Processor Interface
- Constant Interface

The processor interfaces provide the system designer with the ability to dynamically control the parameters within the core.

Core Symbol and Port Descriptions

The Image Noise Reduction core can be configured with three different interface options, each resulting in a slightly different set of ports. The Image Noise Reduction core uses a set of signals that is common to all of the Xilinx Video IP cores called the Xilinx Streaming Video Interface (XSVI). The XSVI signals are shown in Figure 8 and described in Table 2.

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 Noise Reduction IP core uses the following subset 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_NOISE_IN, BUS_TYPE = TARGET, BUS_STD = XSVI
PORT hblank_i =hblank, DIR=I, BUS=XSVI_NOISE_IN
PORT vblank_i =vblank, DIR=I, BUS=XSVI_NOISE_IN
PORT active_video_i =active_video,DIR=I, BUS=XSVI_NOISE_IN
PORT video_data_i =video_data, DIR=I,VEC=[C_DATA_WIDTH-1:0],
BUS=XSVI_NOISE_IN
```

Output Side:

```
BUS_INTERFACE BUS = XSVI_NOISE_OUT, BUS_TYPE = INITIATOR, BUS_STD =
XSVI
PORT hblank_o =hblank, DIR=I, BUS=XSVI_NOISE_OUT
PORT vblank_o =vblank, DIR=I, BUS=XSVI_NOISE_OUT
PORT active_video_o =active_video, DIR=I, BUS=XSVI_NOISE_OUT
PORT video_data_o =video_data, DIR=I,
VEC=[3*C_DATA_WIDTH-1:0],BUS=XSVI_NOISE_OUT
```

The Image Noise Reduction 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 does not provide an option for the filter strength to be changed internally to the system. 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 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.

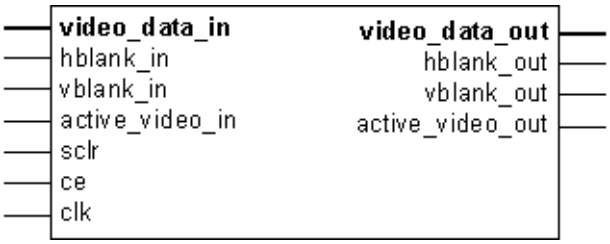

Figure 2-1: **Core Symbol for Constant Interface**

Table 2-1 contains the core port information. Detailed descriptions of the ports are provided in this section as well.

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

Table 2-1: **Port Descriptions for the Constant Interface *(Cont'd)***

Port Name	Port Width	Direction	Description
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	Cr	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 Noise Reduction 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	Cr	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.

General Purpose Processor Interface

The General Purpose Processor Interface exposes the filter strength register as a port. 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 filter strength port has the double-buffer control mechanism described in the previous section to prevent tearing. However, an external register (shadow register) has to be supplied by the user-defined bus interface. Values from this register bank (external to the Image Noise Reduction core) are copied over to the internal registers at the rising edge of vblank_in when bit 1 of the noise_reg00_control register is set to '1'.

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

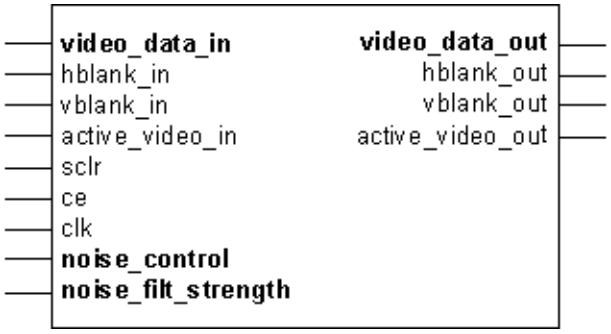

Figure 2-2: **Core Symbol for the General Purpose Processor Interface**

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

Port Name	Port Width	Direction	Description
noise_control	2	IN	Bit 0: Software enable Bit 1: Host processor write done semaphore • 0 indicates host processor actively updating registers • 1 indicates register update completed by host processor
noise_filt_strength	2	IN	Strength of the smoothing filter Possible values are 0, 1, 2, and 3 Select one of four smoothing filters with 0 being the weakest smoothing filter and 3 being the strongest smoothing filter.
noise_status	16	OUT	Bit 7: Timing locked '1' indicates that the timing module of the core has locked on the input timing signals and is generating stable output timing signals

pCore Register Space

The EDK pCore Interface generates AXI4-Lite interface ports in addition to the Streaming Video Signals. The AXI4-Lite bus 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 3. The Streaming Video Interface is described in the previous section (Table 2-1).

Signal details of AXI4-Lite bus interface is shown in Table 2-3.

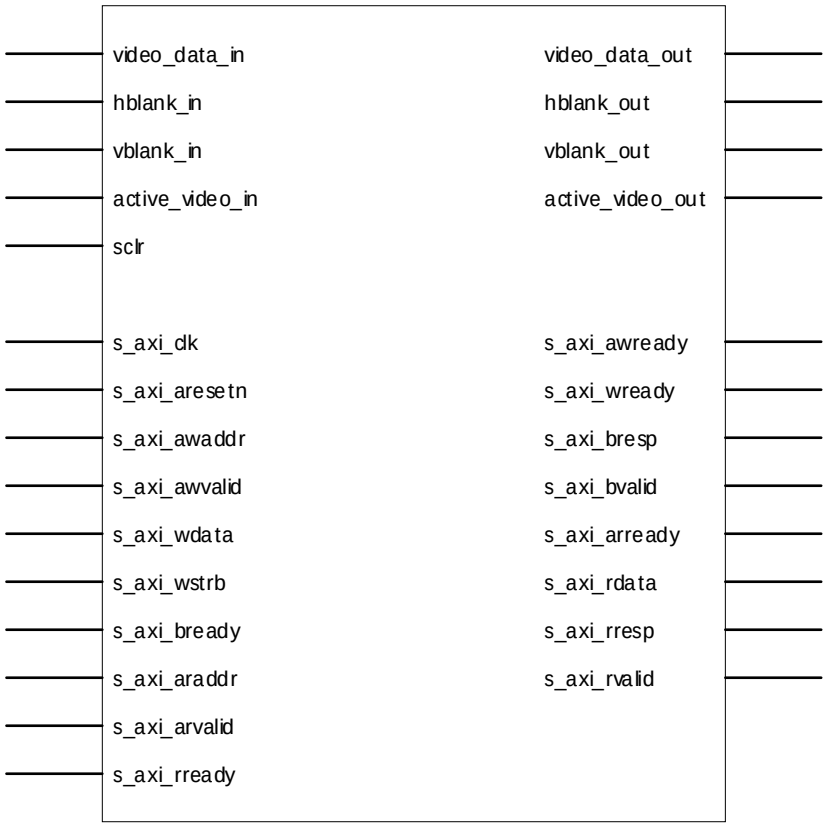

Figure 3: **Core Symbol for the EDK pCore Interface**

Table 2-3: **EDK pCore Signals**

Pin Name	Dir	Width	Description
AXI Global System Signals(1)			
S_AXI_ACLK	I	1	AXI Clock
S_AXI_ARESETN	I	1	AXI Reset, active low
IP2INTC_Irpt	O	1	Interrupt request output
AXI Write Address Channel Signals(1)			
S_AXI_AWADDR	I	[(C_S_AXI_ADDR_WIDTH-1):0]	AXI4-Lite Write Address Bus. The write address bus gives the address of the write transaction.

Table 2-3: **EDK pCore Signals *(Cont'd)***

Pin Name	Dir	Width	Description
S_AXI_AWVALID	I	1	AXI4-Lite Write Address Channel Write Address Valid. This signal indicates that valid write address is available. • 1 = Write address is valid. • 0 = Write address is not valid.
S_AXI_AWREADY	O	1	AXI4-Lite Write Address Channel Write Address Ready. Indicates core is ready to accept the write address. • 1 = Ready to accept address. • 0 = Not ready to accept address.
AXI Write Data Channel Signals(1)			
S_AXI_WDATA	I	[(C_S_AXI_DATA_WIDTH-1):0]	AXI4-Lite Write Data Bus.
S_AXI_WSTRB	I	[C_S_AXI_DATA_WIDTH/8-1:0]	AXI4-Lite Write Strobes. This signal indicates which byte lanes to update in memory.
S_AXI_WVALID	I	1	AXI4-Lite Write Data Channel Write Data Valid. This signal indicates that valid write data and strobes are available. • 1 = Write data/strobes are valid. • 0 = Write data/strobes are not valid.
S_AXI_WREADY	O	1	AXI4-Lite Write Data Channel Write Data Ready. Indicates core is ready to accept the write data. • 1 = Ready to accept data. • 0 = Not ready to accept data.
AXI Write Response Channel Signals(1)			
S_AXI_BRESP(2)	O	[1:0]	AXI4-Lite Write Response Channel. Indicates results of the write transfer. • 00b = OKAY - Normal access has been successful. • 01b = EXOKAY - Not supported. • 10b = SLVERR - Error. • 11b = DECERR - Not supported.

Table 2-3: **EDK pCore Signals *(Cont'd)***

Pin Name	Dir	Width	Description
S_AXI_BVALID	O	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid. • 1 = Response is valid. • 0 = Response is not valid.
S_AXI_BREADY	I	1	AXI4-Lite Write Response Channel Ready. Indicates Master is ready to receive response. • 1 = Ready to receive response. • 0 = Not ready to receive response.
AXI Read Address Channel Signals(1)			
S_AXI_ARADDR	I	[(C_S_AXI_ADDR_WIDTH-1):0]	AXI4-Lite Read Address Bus. The read address bus gives the address of a read transaction
S_AXI_ARVALID	I	1	AXI4-Lite Read Address Channel Read Address Valid. • 1 = Read address is valid. • 0 = Read address is not valid.
S_AXI_ARREADY	O	1	AXI4-Lite Read Address Channel Read Address Ready. Indicates core is ready to accept the read address. • 1 = Ready to accept address. • 0 = Not ready to accept address.
AXI Read Data Channel Signals(1)			
S_AXI_RDATA	O	[(C_S_AXI_DATA_WIDTH-1):0]	AXI4-Lite Read Data Bus.
S_AXI_RRESP(2)	O	[1:0]	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer. • 00b = OKAY - Normal access has been successful. • 01b = EXOKAY - Not supported. • 10b = SLVERR - Error. • 11b = DECERR - Not supported.

Table 2-3: **EDK pCore Signals *(Cont'd)***

Pin Name	Dir	Width	Description
S_AXI_RVALID	O	1	AXI4-Lite Read Data Channel Read Data Valid. This signal indicates that the required read data is available and the read transfer can complete. • 1 = Read data is valid. • 0 = Read data is not valid.
S_AXI_RREADY	I	1	AXI4-Lite Read Data Channel Read Data Ready. Indicates master is ready to accept the read data. • 1 = Ready to accept data. • 0 = Not ready to accept data.

1. The function and timing of these signals are defined in the *AMBA AXI Protocol Version: 2.0 Specification*.
2. For signals S_AXI_RRESP[1:0] and S_AXI_BRESP[1:0], the core does not generate the Decode Error ('11') response. Other responses like '00' (OKAY) and '10' (SLVERR) are generated by the core based upon certain conditions.

EDK pCore Interface

There are multiple imaging applications that include an embedded processor that can dynamically control the parameters within an integrated system. The CORE Generator software can be used to generate the direct pCore interface. The Xilinx MicroBlaze™ processor can be used to control directly the hardware added to an EDK project as a hardware peripheral. This pCore provides a memory-mapped interface for the programmable registers within the core, as described in Table 2-4.

Table 2-4: **EDK pCore Interface Register Descriptions**

Address Offset (hex)	Register Name	Access Type	Default Value (hex)	Description	
BASEADDR +0x800	noise_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	noise_reg01_reset	R/W	0x00000000	Bit 0	Software reset • 0 – Not reset • 1 – Reset

Table 2-4: **EDK pCore Interface Register Descriptions *(Cont'd)***

Address Offset (hex)	Register Name	Access Type	Default Value (hex)	Description	
BASEADDR +0x808	noise_reg02_status	R	0x00000000	Bit 7	Timing locked • 1 - Indicates that the timing module of the core has locked on the input timing signals and is generating stable output timing signals
BASEADDR + 0x80C	noise_reg03_filt_strength	R/W	From GUI	Strength of the smoothing filter Possible values are 0, 1, 2, 3, and 4 1= weakest, 4 = strongest (that is, more noise reduction); setting the filter strength to 0 will bypass the smoothing 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 CORE Generator GUI.

This core supports an enable/disable function. When disabled, the normal operation of the hardware is halted by blocking the propagation of all video signals. This function is controlled by setting the Software Enable, bit 0 of noise_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 noise_reg01_reset (bit 0), which returns the filter strength to its default value, specified through the GUI when the core is instantiated. The core control signals and output are forced to 0 until the software reset bit is deasserted.

The noise_reg03_filt_strength register is double buffered in hardware to ensure no image tearing happens if the filter strength value is 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 with which to update the parameter values. The updated value for the filter strength register is latched into the shadow register immediately after writing, while the actual filter strength used is stored in the working register.

Any reads of registers during operation return the values stored in the shadow 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 noise_reg00_control is set to 1. This semaphore bit helps to prevent changing the filter strength mid-frame.

Chapter 3

Customizing and Generating the Core

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

GUI

The Image Noise Reduction core is easily configured to meet the user's 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 7 shows the main Image Noise Reduction screen.

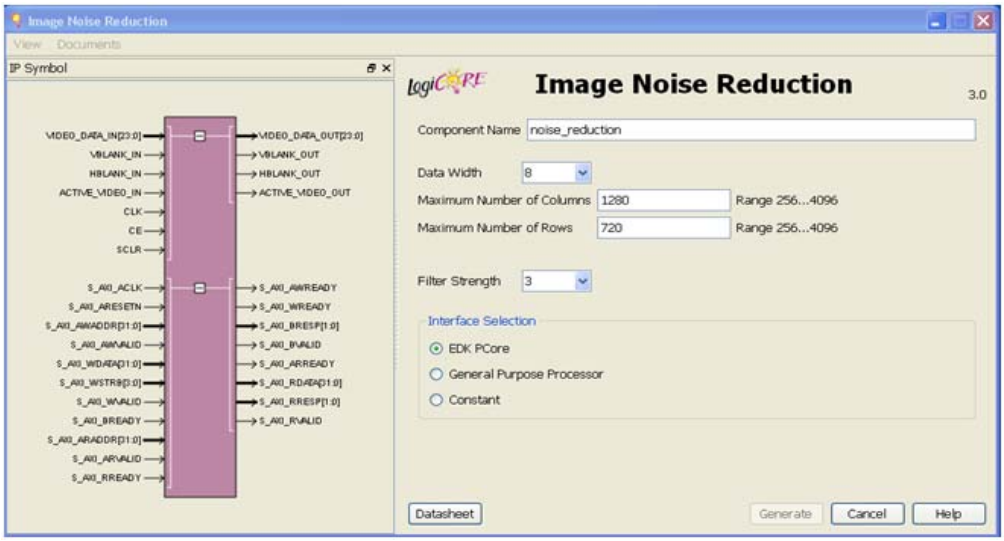

Figure 3-1: **Image Noise Reduction Main Screen**

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 Noise Reduction 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 Noise Reduction instance timing signals that violate the MAX_ROWS constraint leads to data and output timing signal corruption.
- **Filter Strength:** Specifies which of the four smoothing filters to use. The allowed values are 0, 1, 2, and 3. Filter Strength of 0 provides the weakest smoothing, and Filter Strength of 3 provides the strongest smoothing. Therefore, Filter Strength of 3 provides the most noise reduction.
- **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 filter strength can be programmed via a register. Double buffering is used to eliminate tearing of output images. See pCore Register Space 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 filter strength is 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.

Table 3-1: **XCO Parameters**

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
filter_strength	3	0, 1, 2, 3, 4
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.

EDK pCore Files

When the interface selection is set to EDK pCore, CORE Generator will output the core as a pCore that can be easily incorporated into an EDK project. The pCore output consists of a hardware pCore and a software driver. The pCore has the following directory structure in the `<project directory>/<component name>` directory:

- drivers
  - noise_v3_00_a
    - data
    - build
    - example
    - src
- pcores
  - axi_noise_v3_00_a
    - data
    - hdl
      - vhdl

File Details

<project directory>

This is the top-level directory. It contains xco and other assorted files, as shown in Table 3-2.

Table 3-2: **<project directory> Files**

File 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 the CORE Generator software.

<project directory>/<component_name>/pcores/axi_noise_v3_00_a/data

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

< project directory>/<component_name>/pcores/axi_noise_v3_00_a/hdl/vhdl

This directory contains the HDL files that implement the pCore.

< project directory>/<component_name>/drivers/noise_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/noise_v3_00_a/example

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

< project directory>/<component_name>/drivers/noise_v3_00_a/src

This directory contains the source code of the pCore's software driver, as shown in Table 3-3.

Table 3-3: **src Files**

File Name	Description
noise.c	Provides the API access to all features of the device driver.
noise.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, as shown in Table 3-4.

Table 3-4: **GPP Output Files**

Name	Description
<component_name>_readme.txt	Readme file for the core.
<component_name>.ngc	The netlist for the core.
<component_name>.veo <component_name>.vho	The HDL template for instantiating the core.
<component_name>.v <component_name>.vhd	The structural simulation model for the core. It is used for functionally simulating the core.
<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.

Chapter 4

Designing with the Core

This chapter includes guidelines and additional information to make designing with the core easier.

General Design Guidelines

This core performs noise reduction by applying a smoothing filter to the image.

The smoothing filter is applied with a gain K which is dependent on the edge content in the image. Near edges, the gain is low so that the edges have less smoothing applied.

There is a choice of four different smoothing filters. The filters are of increasing strength in terms of the smoothing they provide. There is also an option to bypass the smoothing filter. The filter coefficients and frequency responses are shown in order of increasing strength in Figure 4-1, Figure 4-2, Figure 4-3, and Figure 4-4.

$$\text{Filter } 0 = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 6 & 1 \\ 1 & 2 & 1 \end{bmatrix}$$

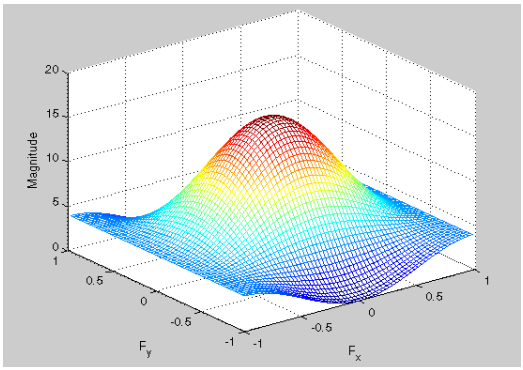

Figure 4-1: **Coefficients and Frequency Response for Filter Strength 0**

$$\text{Filter } 1 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

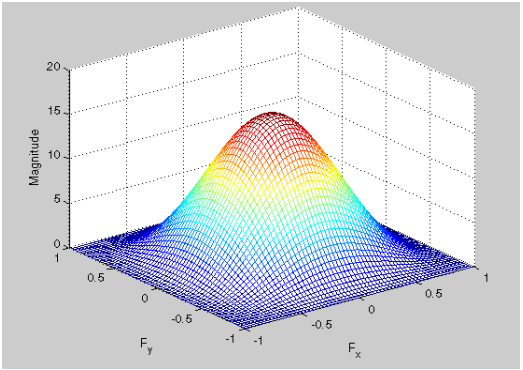

Figure 4-2: **Coefficients and Frequency Response for Filter Strength 1**

$$\text{Filter 2} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 2 & 4 & 2 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

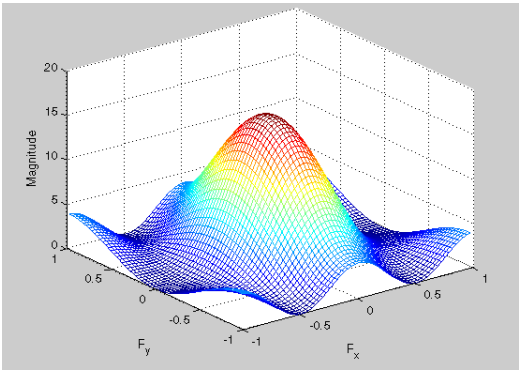

Figure 4-3: **Coefficients and Frequency Response for Filter Strength 2**

$$\text{Filter 3} = \begin{bmatrix} 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & 2 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \end{bmatrix}$$

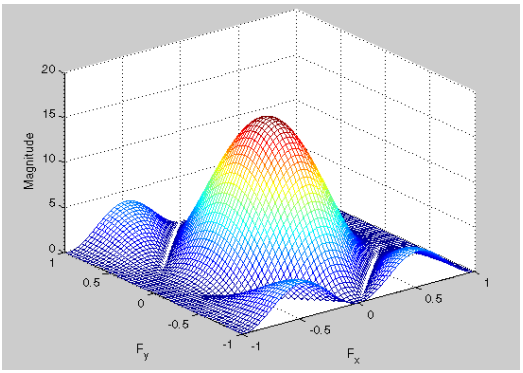

Figure 4-4: **Coefficients and Frequency Response for Filter Strength 3**

Clocking

The Image Noise Reduction core has one clock (`clk`) that is used to clock the entire core. This includes the AXI interface and the core logic.

Control Signals and Timing

The propagation delay of the Image Noise Reduction core is one full scan line and 18 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.

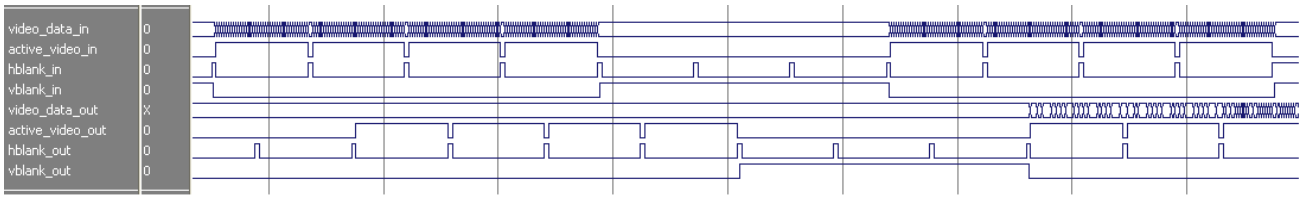

Figure 4-5: **Timing Example**

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.

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.

Resets

The Image Noise Reduction 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 Noise Reduction core, the register interface is compliant with the AXI4-Lite interface.

Constraining the Core

This chapter contains the applicable constraints for the Image Noise Reduction core.

Required Constraints

There are no required constraints for this core.

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 lower power devices.

Clock Frequencies

There are no specific clock frequency requirements for this core other than the Maximum Frequency discussed in the Performance section.

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

This chapter contains details about the demonstration test bench included with the core.

Demonstration Test Bench

A demonstration test bench is provided as an introductory package that enables core users to observe the core generated by the CORE Generator tool 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.

The latest version of the test bench is available for download on the Image Noise Reduction product page.

Directory and File Contents

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

- Expected: Contains the pre-generated expected/golden data used by the test bench to compare actual output data.
- Stimuli: Contains the pre-generated input data used by the test bench 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 the CORE Generator tool. A new netlist can be generated using the .xco file in the CORE Generator tool.
- tb_src: Contains the top-level test bench design. This directory also contains other packages used by the test bench.
- isim_wave.wcfg: Waveform configuration for ISIM.
- mti_wave.do: Waveform configuration for Mentor Graphics 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.

Simulation

To begin a simulation using ModelSim for Linux, type **source run_mti.sh** from the console.

To begin a simulation using ModelSim for Windows, double-click the `run_mti.bat`.

To begin a simulation using iSim for Linux, type **source run_isim.sh** from the console.

To begin a simulation using iSim for Windows, double-click the `run_isim.bat`.

XILINX®

Appendix A

Verification, Compliance, and Interoperability

The Image Noise Reduction core has been verified with extensive simulation and hardware verification.

Simulation

A highly parameterizable test bench used to test the Image Noise Reduction core. Testing includes the following:

- Register accesses
- Testing of all the filter strengths
- Testing of various data widths

Hardware Testing

The Image Noise Reduction 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

This appendix describes migrating from older versions of the IP to the current IP release.

Migrating to the EDK pCore AXI4-Lite Interface

The Image Noise Reduction v3.0 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 UG761, *Xilinx AXI Reference Guide*.

Parameter Changes in the XCO File

There are no parameter changes in the XCO file.

Port Changes

Other than an AXI4-Lite interface in place of the PLB, there are no port changes.

Functionality Changes

There are no functionality changes to the core.

Appendix C

Debugging

This chapter contains information to help with debugging the core.

Evaluation Core Timeout

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

XILINX®

Appendix D

Application Software Development

This appendix contains details about the software API provided with the core.

Reading and Writing pCore Registers

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

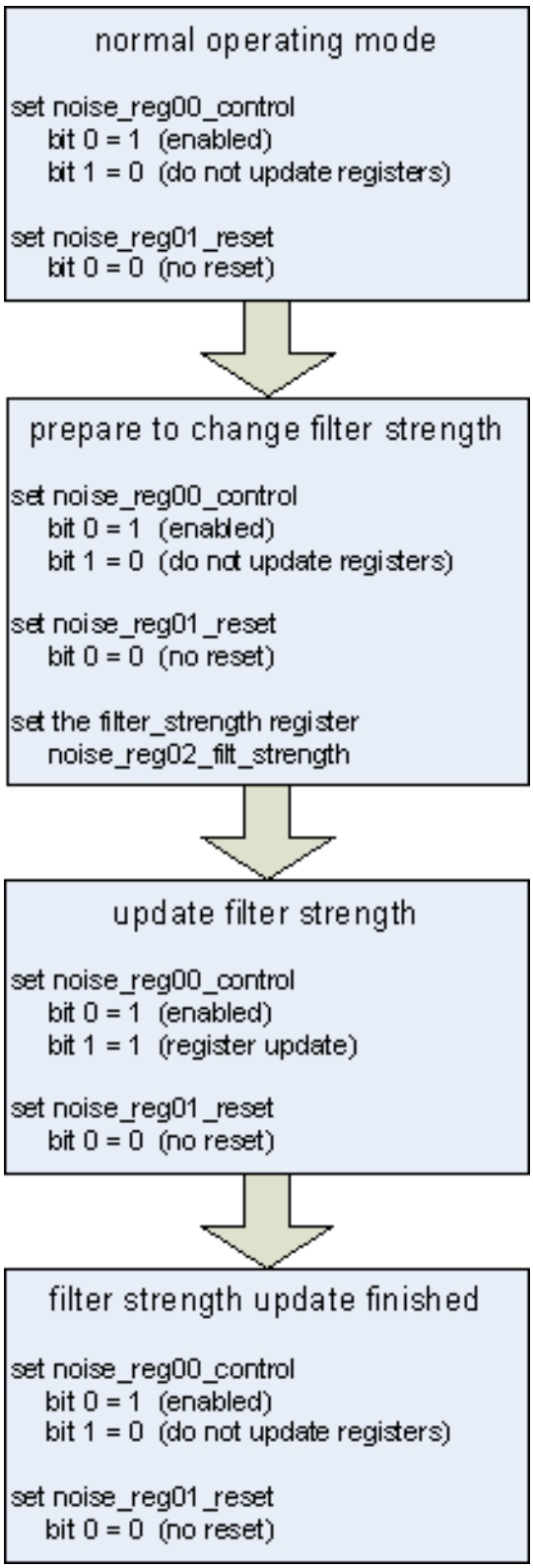

Figure D-1: **Image Noise Reduction Programming Flow Chart**

EDK pCore API Functions

The software API is provided to allow easy access to the Image Noise Reduction pCore's registers defined in Table 2-4, page 17. To use the API functions provided, the following two header files must be included in the user C code:

```
#include “noise.h”
#include “xparameters.h”
```

The hardware settings of your system, including the base address of your Image Noise Reduction core, are defined in the xparameters.h file. The noise.h file contains the macro function definitions for controlling the Image Noise Reduction 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 noise_v3_00_a/example subfolder. This file is a sample C program that demonstrates how to use the Image Noise Reduction pCore API.

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

- NOISE_Enable(uint32 BaseAddress);
  - This macro enables an Image Noise Reduction instance.
  - BaseAddress is the Xilinx EDK base address of the Noise Reduction core (from xparameters.h).
- NOISE_Disable(uint32 BaseAddress);
  - This macro disables an Noise Reduction instance.
  - BaseAddress is the Xilinx EDK base address of the Noise Reduction core (from xparameters.h).
- NOISE_Reset(uint32 BaseAddress);
  - This macro resets an Noise Reduction 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 NOISE_ClearReset() is called.
  - BaseAddress is the Xilinx EDK base address of the Noise Reduction core (from xparameters.h)
- NOISE_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 Noise Reduction core (from xparameters.h).

Each software register defined in Table 2-4, page 17 has a constant defined in noise.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 = NOISE_ReadReg(XPAR_NOISE_0_BASEADDR,
NOISE_REG03_FILT_STRENGTH);
```

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

- NOISE_ReadReg(uint32 BaseAddress, uint32 RegOffset)
  - Read the given register.

- BaseAddress is the Xilinx EDK base address of the Noise Reduction core (from xparameters.h).
- RegOffset is the register offset of the register (defined in Table 2-4, page 17).

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

```
NOISE_WriteReg(XPAR_NOISE_0_BASEADDR, NOISE_REG03_FILT_STRENGTH, 0);
```

The definition of this macro is:

- NOISE_WriteReg(uint32 BaseAddress, uint32 RegOffset, uint32 Data)
  - Write the given register.
  - BaseAddress is the Xilinx EDK base address of the Noise Reduction 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.
- NOISE_RegUpdateEnable(uint32 BaseAddress);
  - Calling RegUpdateEnable causes the Noise Reduction 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 Noise Reduction core is enabled.
  - BaseAddress is the Xilinx EDK base address of the Noise Reduction core (from xparameters.h)
- NOISE_RegUpdateDisable(uint32 BaseAddress);
  - Disabling the Register Update prevents the Noise Reduction 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 Noise Reduction core is enabled.
  - BaseAddress is the Xilinx EDK base address of the Noise Reduction core (from xparameters.h)

XILINX®

Appendix E

C Model Reference

This document introduces the bit accurate C model for the Xilinx® LogiCORE™ IP Image Noise Reduction core, which has been developed primarily for system modeling.

Features

- Bit accurate with the Image Noise Reduction core
- Statically linked library (.lib, .o, .obj – Windows)
- Dynamically linked library (.so – Linux)
- Available for 32 and 64-bit for both Windows and Linux
- Supports all features of the Image Noise Reduction core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C code is provided to show how to use the function
- Example application C code wrapper file supports 8-bit BMP and YUV

Overview

The Xilinx LogiCORE IP Image Noise Reduction core has a bit accurate C model for 32 and 64-bit Windows and Linux platforms. The model has an interface consisting of a set of C functions, which reside in a statically link library (shared library). Full details of the interface are provided in Chapter 3, Using the C Model. An example piece of C code is provided to show how to call the model.

The model is bit accurate, as it produces exactly the same output data as the core on a frame-by-frame basis. However, the model is not cycle accurate, as it does not model the core's latency or its interface signals.

The latest version of the model is available for download on the Xilinx™ LogiCORE IP Image Noise Reduction web page at:

http://www.xilinx.com/products/intellectual-property/EF-DI-IMG-NOISE.htm

User Instructions

This section contains information on using the C Model.

Unpacking and Model Contents

Unzip the `v_noise_v3_0_bitacc_model.zip` file, containing the bit-accurate models for the Image Noise Reduction core. This creates the directory structure and files in Table E-1.

Table E-1: **Directory Structure and Files of the Image Noise Reduction Bit-Accurate C Model**

File Name	Contents
README.txt	Release notes
pg011_v_noise.pdf	*LogiCORE IP Image Noise Reduction Product Guide*
v_noise_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
/lin	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms
libIp_v_noise_v3_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_noise_v3_0_bitacc_cmodel.so
/lin64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms
libIp_v_noise_v3_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_noise_v3_0_bitacc_cmodel.so
/nt	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms.
libIp_v_noise_v3_0_bitacc_cmodel.lib	Precompiled library file for win32 compilation
libIp_v_noise_v3_0_bitacc_cmodel.dll	Precompiled library file for win32 compilation
/nt64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms.
libIp_v_noise_v3_0_bitacc_cmodel.lib	Precompiled library file for win64 compilation
libIp_v_noise_v3_0_bitacc_cmodel.dll	Precompiled library file for win64 compilation

Installation

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

- libIp_v_noise_v3_0_bitacc_cmodel.so
- libstlport.so.5.1

Software Requirements

The Image Noise Reduction C models are 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-bit and 64-bit Linux	GCC 4.1.1
32-bit and 64-bit Windows	Microsoft Visual Studio 2008

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_noise_v3_0_bitacc_cmodel.h` file.

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

```
struct  xilinx_ip_v_noise_v3_0_generics noise_generics;
struct  xilinx_ip_v_noise_v3_0_inputs   noise_inputs;
struct  xilinx_ip_v_noise_v3_0_outputs  noise_outputs;
```

The declaration of these structures is in the `v_noise_v3_0_bitacc_cmodel.h` file.

Table E-3 lists the generic parameters taken by the Image Noise Reduction 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_noise_v3_0_get_default_generics(&noise_generics)` initializes the generics structure with the defaults, listed in Table E-3.

The smoothing filter selection can also be set dynamically through the pCore and General Purpose Processor interfaces. This value is passed as an input to the core, along with the actual test image, or video sequence (see 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]
filt_strength	int	3	0, 1, 2, 3, 4	Smoothing filter selection

1. For the description of the input structure, see Initializing the Image Noise Reduction Input Video Structure.

The structure `noise_inputs` defines the values of run time parameters and the actual input image.

Calling `xilinx_ip_v_noise_v3_0_get_default_inputs(&noise_generics, &noise_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_noise_v3_0_bitacc_simulate(
struct xilinx_ip_v_noise_v3_0_generics* generics,
struct xilinx_ip_v_noise_v3_0_inputs*   inputs,
struct xilinx_ip_v_noise_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_noise_v3_0_destroy(
struct xilinx_ip_v_noise_v3_0_inputs *input,
struct xilinx_ip_v_noise_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 Noise Reduction Input and Output Video Structure

Input images or video streams can be provided to the Image Noise Reduction 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.

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

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 Noise Reduction C model supports the following mode: FORMAT_C444.

Initializing the Image Noise Reduction 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 `yuv_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_yuv8p(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 file (BIN file)
out_file    : path/name of the output file (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 Noise Reduction 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 /lin or /lin64 directory to the root directory:

   `libstlport.so.5.1`

   `libIp_v_noise_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_noise_v3_0_bitacc_cmodel -Wl,-rpath,.


   gcc –m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
   run_bitacc_cmodel -L. -lIp_v_noise_v3_0_bitacc_cmodel -Wl,-rpath,.
   ```

Windows (32-bit and 64-bit)

The precompiled library `v_noise_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 Console Application project.

2. As existing items, add:
   a. `libIp_v_noise_v3_0_bitacc_cmodel.lib` to the Resource Files folder of the project
   b. `run_bitacc_cmodel.c` and `parsers.c` to the Source Files folder of the project
   c. `v_noise_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 an 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.

References

These documents provide supplemental material useful with this user guide:

- *AMBA AXI4-Stream Protocol Specification*
- *AMBA AXI Protocol Version: 2.0 Specification*
- UG761, *Xilinx AXI Reference Guide*
- DS768, *Xilinx AXI Interconnect IP Data Sheet*

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 Noise Reduction core is provided under 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 Image Noise Reduction product page.

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