LogiCORE IP Image Noise Reduction v4.00.a

Product Guide

PG011 April 24, 2012

Table of Contents

Chapter 1: Overview

Feature Summary 6
Applications 6
Licensing 7

Chapter 2: Product Specification

Standards Compliance 8
Performance 8
Resource Utilization 9
Core Interfaces and Register Space 11
Common Interface Signals 12
Data Interface 13
Control Interface 15
Register Space 17

Chapter 3: Customizing and Generating the Core

GUI 23
Parameter Values in the XCO File 25
Output Generation 26

Chapter 4: Designing with the Core

General Design Guidelines 27
Clock, Enable, and Reset Considerations 29
System Considerations 31

Chapter 5: Constraining the Core

Required Constraints 33

Device, Package, and Speed Grade Selections 33
Clock Frequencies 33
Clock Management 33
Clock Placement 33
Banking 34
Transceiver Placement 34
I/O Standard and Placement 34

Chapter 6: Detailed Example Design

Demonstration Test Bench 35
Test Bench Structure 35
Running the Simulation 36
Directory and File Contents 36

Appendix A: Verification, Compliance, and Interoperability

Simulation 38
Hardware Testing 38
Interoperability 39

Appendix B: Migrating

Appendix C: Debugging

Bringing up the AXI4-Lite Interface 41
Bringing up the AXI4-Stream Interfaces 42
Debugging Features 43
Evaluation Core Timeout 45

Appendix D: Application Software Development

Programmer's Guide 46

Appendix E: C Model Reference

Features 49
Overview 49

User Instructions 50
Using the C Model 51
C Model Example Code 56

Appendix F: Additional Resources

Xilinx Resources 58
References 58
Technical Support 58
Ordering Information. 59
Revision History 59
Notice of Disclaimer. 59

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

- In-system update of smoothing filters
- YCbCr 4:4:4 input and output
- AXI4-Stream data interfaces
- Optional AXI4-Lite control interface
- Supports 8, 10, and 12 bits per color component input and output
- Built-in, optional bypass and test-pattern generator mode
- Built-in, optional throughput monitors
- Supports spatial resolutions from 32x32 up to 7680x7680
    - Supports 1080P60 in all supported device families
    - Supports 4kx2k @ 24 Hz in supported high performance devices

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq™-7000, Artix™-7, Virtex®-7, Kintex®-7, Virtex-6, Spartan®-6
Supported User Interfaces	AXI4-Lite, AXI4-Stream
Resources	See Table 2-1 through Table 2-6.
Provided with Core	
Documentation	Product Guide
Design Files	NGC netlist, Encrypted HDL
Example Design	Not Provided
Test Bench	Verilog [2]
Constraints File	Not Provided
Simulation Models	VHDL or Verilog Structural, C-Model [2]
Tested Design Tools	
Design Entry Tools	CORE Generator™ tool, Platform Studio (XPS)
Simulation[3]	Mentor Graphics ModelSim, Xilinx ISim
Synthesis Tools	Xilinx Synthesis Technology (XST)
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/intellectual-property/EF-DI-IMG-NOISE.htm.
3. For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.

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 edges have less smoothing applied.

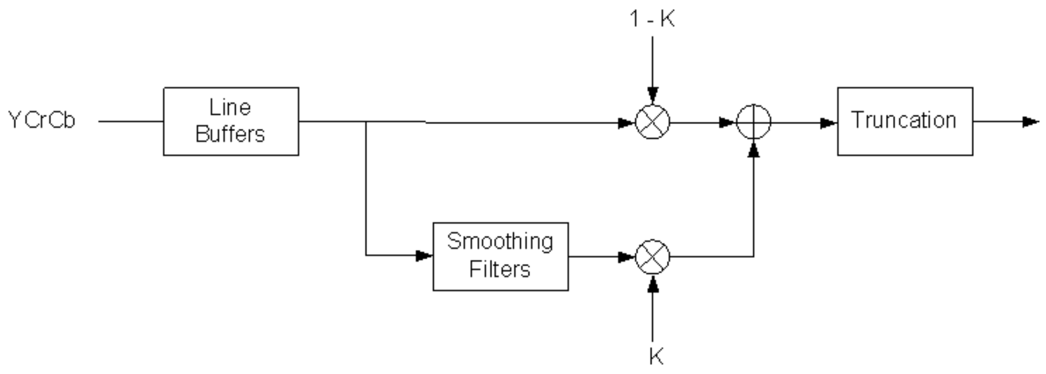


Figure 1-1: **Image Noise Reduction**

Feature Summary

The core is capable of a maximum resolution of 7680 columns by 7680 rows with 8, 10, or 12 bits per pixel. In addition, the Image Noise Reduction core supports bandwidths necessary for high-definition (1080p60) resolutions in all Xilinx FPGA device families. Higher resolutions are supported in Xilinx high-performance device families. the core is configured and instantiated from CORE Generator or EDK tools. Core functionality can be controlled dynamically with an optional AXI4-Lite interface.

Applications

- Pre-processing block for image sensors
- Video surveillance
- Industrial imaging
- Video conferencing

- Machine vision
- Other imaging applications

Licensing

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The core may be generated using either the Xilinx ISE CORE Generator tool or Embedded Edition software (EDK). For full access to all core functionality in simulation and in hardware, you must purchase a license for the core. Please contact your local Xilinx sales representative for information on pricing and availability of Xilinx LogiCORE IP.

For more information, please visit the LogiCORE IP Image Noise Reduction product page.

Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE modules and software, please contact your local Xilinx sales representative.

Product Specification

This chapter details the performance, ports and registers for the Image Noise Reduction core.

Standards Compliance

The Image Noise reduction core is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. See "Video IP: AXI Feature Adoption" in UG761, *Xilinx AXI Reference Guide* for additional information.

Performance

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

Maximum Frequencies

This section contains the 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 FPGA device, using a different version of Xilinx tools and other factors. Refer to Table 2-1 through Table 2-6 for device-specific information.

Latency

The propagation delay of the Image Noise Reduction core is one full scan line and 26 video clock cycles.

Throughput

The Image Noise Reduction core produces one output pixel per input sample.

The core supports bidirectional data throttling between its AXI4-Stream Slave and Master interfaces. If the slave side data source is not providing valid data samples (`s_axis_video_tvalid` is not asserted), the core cannot produce valid output samples after its internal buffers are depleted. Similarly, if the master side interface is not ready to accept valid data samples (`m_axis_video_tready` is not asserted) the core cannot accept valid input samples once its buffers become full.

If the master interface is able to provide valid samples (`s_axis_video_tvalid` is high) and the slave interface is ready to accept valid samples (`m_axis_video_tready` is high), typically the core can process one sample and produce one pixel per `ACLK` cycle.

However, at the end of each scan line the core flushes internal pipelines for 26 clock cycles, during which the `s_axis_video_tready` is de-asserted signaling that the core is not ready to process samples. Also at the end of each frame the core flushes internal line buffers for 1 scan line, during which the `s_axis_video_tready` is de-asserted signaling that the core is not ready to process samples.

When the core is processing timed streaming video (which is typical for image sensors), the flushing periods coincide with the blanking periods therefore do not reduce the throughput of the system.

When the core is processing data from a video source which can always provide valid data, e.g. a frame buffer, the throughput of the core can be defined as follows:

$$R_{MAX} = f_{ACLK} \times \frac{ROWS}{ROWS+1} \times \frac{COLS}{COLS+26} \quad \textit{Equation 2-1}$$

In numeric terms, 1080P/60 represents an average data rate of 124.4 MPixels/second (1080 rows x 1920 columns x 60 frames / second), and a burst data rate of 148.5 MPixels/sec.

To ensure that the core can process 124.4 MPixels/second, it needs to operate minimally at:

$$f_{ACLK} = R_{MAX} \times \frac{ROWS+1}{ROWS} \times \frac{COLS+26}{COLS} = 124.4 \times \frac{1081}{1080} \times \frac{1946}{1920} = 126.2 \quad \textit{Equation 2-2}$$

Resource Utilization

For an accurate measure of the usage of primitives, slices, and CLBs for a particular instance, check the **Display Core Viewer after Generation** check box in the CORE Generator interface.

The information presented in Table 2-1 through Table 2-6 is a guide to the resource utilization and maximum clock frequency of the Image Noise Reduction core for Artix-7, Zynq-7000, Virtex-7, Kintex-7, Virtex-6, and Spartan-6 FPGA families. This core does not use any dedicated I/O or CLK resources. The design was tested using ISE® v14.1 tools with default tool options for characterization data. The design was tested with the AXI4-Lite interface, INTC_IF and the Debug Features disabled. By default, the maximum number of pixels per scan line was set to 1920, active pixels per scan line was set to 1920.

Table 2-1: **Zynq-7000 Resource Usage**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36/18	DSP48E1s	Fmax (MHz)
8	1277	1114	1200	3/0	3	163
10	1392	1293	1440	3/1	3	263
12	1697	1483	1680	4/0	3	239

Device: XC7Z030-1FFG676C (ADVANCED 1.01d 2012-04-02)

Table 2-2: **Artix-7 Resource Usage**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36/18	DSP48E1s	Fmax (MHz)
8	1226	1105	1198	3/0	3	180
10	1472	1292	1438	3/1	3	180
12	1717	1473	1678	4/0	3	173

Device: XC7A100T-1FGG484C (ADVANCED 1.03j 2012-04-02)

Table 2-3: **Virtex-7 Resource Usage**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36/18	DSP48E1s	Fmax (MHz)
8	1261	1105	1200	3/0	3	273
10	1464	1290	1440	3/1	3	263
12	1655	1486	1680	4/0	3	253

Device: XC7V585T-1FFG1157C (ADVANCED 1.04j 2012-04-02)

Table 2-4: **Kintex-7 Resource Usage**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36/18	DSP48E1s	Fmax (MHz)
8	1155	1105	1200	3/0	3	246
10	1400	1296	1440	3/1	3	272
12	1700	1482	1680	4/0	3	263

Device: XC7K70T-1FBG484C (ADVANCED 1.04c 2012-04-02)

Table 2-5: **Virtex-6 Resource Usage**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36/18	DSP48E1s	Fmax (MHz)
8	1195	1108	1198	3/0	3	239
10	1405	1300	1438	3/1	3	262
12	1623	1474	1678	4/0	3	255

Device: XC6VLX75T-1FF484C (PRODUCTION 1.17 2012-04-02)

Table 2-6: **Spartan-6 Resource Usage**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 16/8	DSP48A1s	Fmax (MHz)
8	1259	1089	1290	5/1	3	169
10	1434	1272	1548	7/0	3	169
12	1665	1464	1806	8/0	3	164

Device: XC6SLX25-2FGG484C (PRODUCTION 1.21 2012-04-02)

Core Interfaces and Register Space

The Image Noise Reduction core uses industry standard control and data interfaces to connect to other system components. The following sections describe the various interfaces available with the core. Figure 2-1 illustrates an I/O diagram of the Image Noise Reduction core. Some signals are optional and not present for all configurations of the core. The AXI4-Lite interface and the `IRQ` pin are present only when the core is configured via the GUI with an AXI4-Lite control interface. The `INTC_IF` interface is present only when the core is configured via the GUI with the INTC interface enabled.

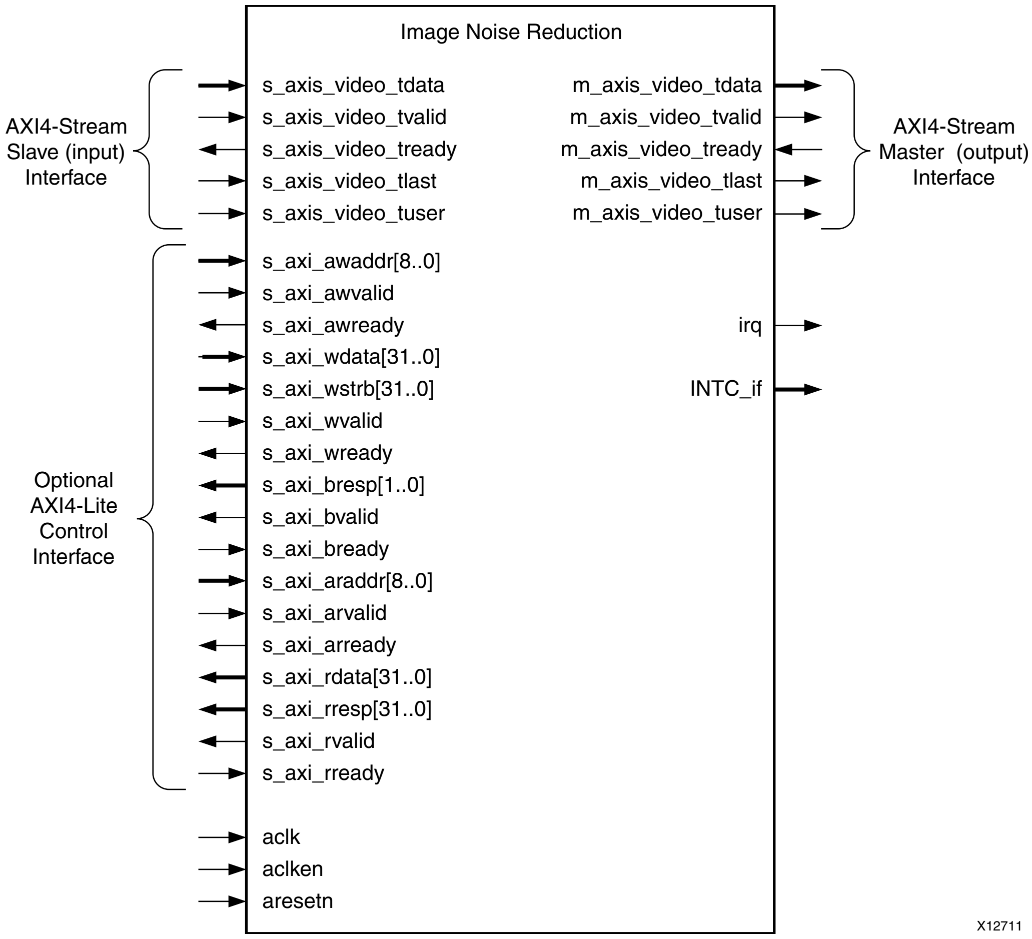


Figure 2-1: **Image Noise Reduction Top-Level Signaling Interface**

Common Interface Signals

Table 2-7 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-7: **Common Interface Signals**

Signal Name	Direction	Width	Description
ACLK	In	1	Clock
ACLKEN	In	1	Clock Enable
ARESETn	In	1	Active low synchronous
INTC_IF	Out	9	Optional External Interrupt Controller Interface. Available only when INTC_IF is selected on GUI.
IRQ	Out	1	Optional Interrupt Request Pin. Available only when AXI4-Lite interface is selected on GUI.

The ACLK, ACLKEN and ARESETn signals are shared between the core, the AXI4-Stream data interfaces, and the AXI4-Lite control interface. Refer to Interrupt Subsystem for a description of the INTC_IF and IRQ pins.

ACLK

All signals, including the AXI4-Stream and AXI4-Lite component interfaces, must be synchronous to the core clock signal ACLK. All interface input signals are sampled on the rising edge of ACLK. All output signal changes occur after the rising edge of ACLK.

ACLKEN

The ACLKEN pin is an active-high, synchronous clock-enable input pertaining to both the AXI4-Stream and AXI4-Lite interfaces. Setting ACLKEN low (de-asserted) halts the operation of the core despite rising edges on the ACLK pin. Internal states are maintained, and output signal levels are held until ACLKEN is asserted again. When ACLKEN is de-asserted, core inputs are not sampled, except ARESETn, which supersedes ACLKEN.

ARESETn

The ARESETn pin is an active-low, synchronous reset input pertaining to both the AXI4-Stream and AXI4-Lite interfaces. ARESETn supersedes ACLKEN, and when set to 0, the core resets at the next rising edge of ACLK even if ACLKEN is de-asserted.

Data Interface

The Image Noise Reduction core receives and transmits data using AXI4-Stream interfaces that implement a video protocol as defined in the *AXI Reference Guide (UG761)*, Video IP: AXI Feature Adoption section.

AXI4-Stream Signal Names and Descriptions

Table 2-8 describes the AXI4-Stream signal names and descriptions.

Table 2-8: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
s_axis_video_tdata	In	24, 32, 40	Input Video Data
s_axis_video_tvalid	In	1	Input Video Valid Signal
s_axis_video_tready	Out	1	Input Ready
s_axis_video_tuser	In	1	Input Video Start Of Frame
s_axis_video_tlast	In	1	Input Video End Of Line
m_axis_video_tdata	Out	24,32,40	Output Video Data
m_axis_video_tvalid	Out	1	Output Valid
m_axis_video_tready	In	1	Output Ready
m_axis_video_tuser	Out	1	Output Video Start Of Frame
m_axis_video_tlast	Out	1	Output Video End Of Line

Video Data

The AXI4-Stream interface specification restricts TDATA widths to integer multiples of 8 bits. Therefore, 10 and 12 bit sensor data must be padded with zeros on the MSB to form a 32- or 40-bit wide vector before connecting to `s_axis_video_tdata`. Padding does not affect the size of the core.

Similarly, YCbCr data on the Image Noise Reduction output `m_axis_video_tdata` is packed and padded to multiples of 8 bits as necessary, as seen in Figure 2-2. Zero padding the most significant bits is only necessary for 10 and 12 bit wide data.

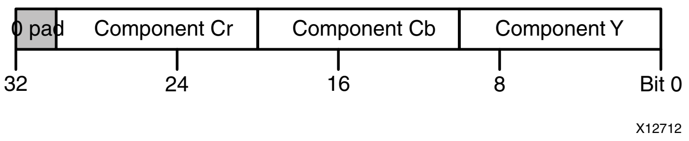


Figure 2-2: **YCbCr Data Encoding on m_axis_video_tdata**

READY/VALID Handshake

A valid transfer occurs whenever READY, VALID, ACLKEN, and ARESETn are high at the rising edge of ACLK, as seen in Figure 2-9. During valid transfers, DATA only carries active video data. Blank periods and ancillary data packets are not transferred via the AXI4-Stream video protocol.

Guidelines on Driving s_axis_video_tvalid

Once s_axis_video_tvalid is asserted, no interface signals (except the Image Noise Reduction core driving s_axis_video_tready) may change value until the transaction completes (s_axis_video_tready, s_axis_video_tvalid ACLKEN high on the rising edge of ACLK). Once asserted, s_axis_video_tvalid may only be de-asserted after a transaction has completed. Transactions may not be retracted or aborted. In any cycle following a transaction, s_axis_video_tvalid can either be de-asserted or remain asserted to initiate a new transfer.

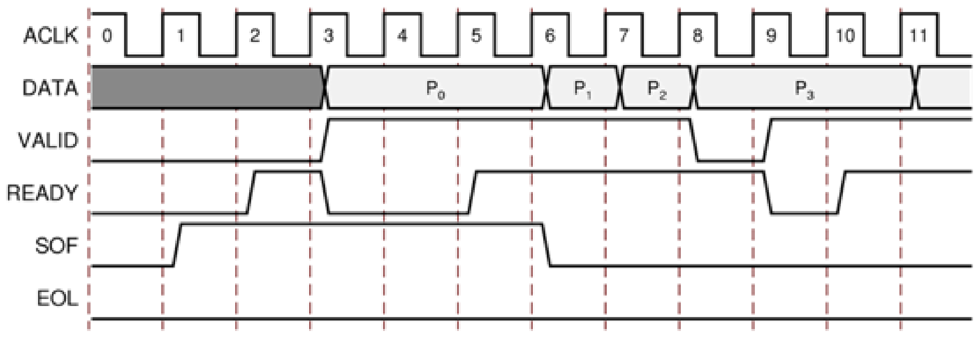


Figure 2-3: **Example of READY/VALID Handshake, Start of a New Frame**

Guidelines on Driving m_axis_video_tready

The m_axis_video_tready signal may be asserted before, during or after the cycle in which the Image Noise Reduction core asserted m_axis_video_tvalid. The assertion of m_axis_video_tready may be dependent on the value of m_axis_video_tvalid. A slave that can immediately accept data qualified by m_axis_video_tvalid, should pre-assert its m_axis_video_tready signal until data is received. Alternatively, m_axis_video_tready can be registered and driven the cycle following VALID assertion. It is recommended that the AXI4-Stream slave should drive READY independently, or pre-assert READY to minimize latency.

Start of Frame Signals - m_axis_video_tuser, s_axis_video_tuser

The Start-Of-Frame (SOF) signal, physically transmitted over the AXI4-Stream TUSER0 signal, marks the first pixel of a video frame. The SOF pulse is 1 valid transaction wide, and must coincide with the first pixel of the frame, as seen in Figure 2-3. SOF serves as a frame synchronization signal, which allows downstream cores to re-initialize, and detect the first pixel of a frame. The SOF signal may be asserted an arbitrary number of ACLK cycles before the first pixel value is presented on DATA, as long as a VALID is not asserted.

End of Line Signals - m_axis_video_tlast, s_axis_video_tlast

The End-Of-Line signal, physically transmitted over the AXI4-Stream TLAST signal, marks the last pixel of a line. The EOL pulse is 1 valid transaction wide, and must coincide with the last pixel of a scan-line, as seen in Figure 2-4.

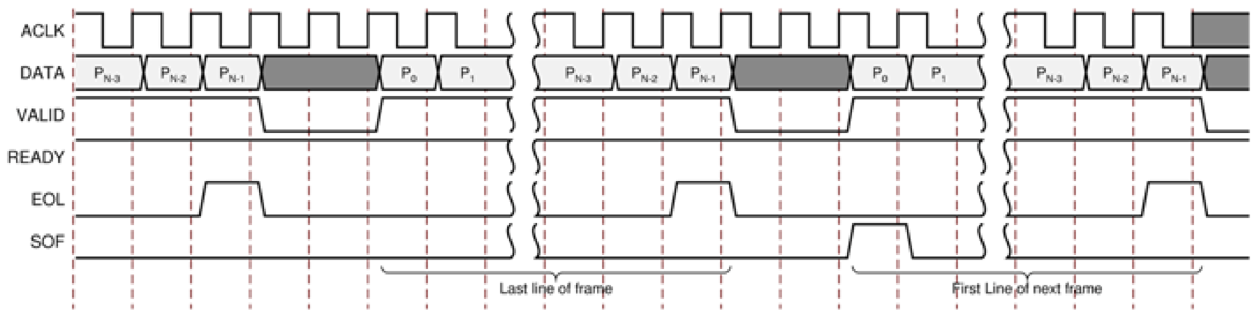


Figure 2-4: **Use of EOL and SOF Signals**

Control Interface

When configuring the core, the user has the option to add an AXI4-Lite register interface to dynamically control the behavior of the core. The AXI4-Lite slave interface facilitates integrating the core into a processor system, or along with other video or AXI4-Lite compliant IP, connected via AXI4-Lite interface to an AXI4-Lite master. In a static configuration with a fixed set of parameters (constant configuration), the core can be instantiated without the AXI4-Lite control interface, which reduces the core Slice footprint.

Constant Configuration

The constant configuration caters to users who will interface the core to a particular video stream with a known, stationary resolution and filter strength. In constant configuration the image resolution (number of active pixels per scan line and the number of active scan lines per frame), and the filter strength is hard coded into the core via the Image Noise

Reduction core GUI. Since there is no AXI4-Lite interface, the core is not programmable, but can be reset, enabled, or disabled using the ARESETn and ACLKEN ports.

AXI4-Lite Interface

The AXI4-Lite interface allows a user to dynamically control parameters within the core. Core configuration can be accomplished using an AXI4-Stream master state machine, or an embedded ARM or soft system processor such as MicroBlaze.

The Image Noise Reduction core can be controlled via the AXI4-Lite interface using read and write transactions to the Image Noise Reduction register space.

Table 2-9: **AXI4-Lite Interface Signals**

Signal Name	Direction	Width	Description
`s_axi_lite_awvalid`	In	1	AXI4-Lite Write Address Channel Write Address Valid.
`s_axi_lite_awread`	Out	1	AXI4-Lite Write Address Channel Write Address Ready. Indicates DMA ready to accept the write address.
`s_axi_lite_awaddr`	In	32	AXI4-Lite Write Address Bus
`s_axi_lite_wvalid`	In	1	AXI4-Lite Write Data Channel Write Data Valid.
`s_axi_lite_wready`	Out	1	AXI4-Lite Write Data Channel Write Data Ready. Indicates DMA is ready to accept the write data.
`s_axi_lite_wdata`	In	32	AXI4-Lite Write Data Bus
`s_axi_lite_bresp`	Out	2	AXI4-Lite Write Response Channel. Indicates results of the write transfer.
`s_axi_lite_bvalid`	Out	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid.
`s_axi_lite_bready`	In	1	AXI4-Lite Write Response Channel Ready. Indicates target is ready to receive response.
`s_axi_lite_arvalid`	In	1	AXI4-Lite Read Address Channel Read Address Valid
`s_axi_lite_arready`	Out	1	Ready. Indicates DMA is ready to accept the read address.
`s_axi_lite_araddr`	In	32	AXI4-Lite Read Address Bus
`s_axi_lite_rvalid`	Out	1	AXI4-Lite Read Data Channel Read Data Valid
`s_axi_lite_rready`	In	1	AXI4-Lite Read Data Channel Read Data Ready. Indicates target is ready to accept the read data.
`s_axi_lite_rdata`	Out	32	AXI4-Lite Read Data Bus
`s_axi_lite_rresp`	Out	2	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer.

Register Space

The standardized Xilinx Video IP register space is partitioned to control-, timing-, and core specific registers. The Image Noise Reduction core uses only one timing related register, ACTIVE_SIZE (0x0020), which allows specifying the input frame dimensions. Also, the core has only one core-specific register, FILT_STRENGTH (0x0100) which allows specifying the strength of the smoothing filter, as described in FILT_STRENGTH (0x0100) Register.

Table 2-10: **Register Names and Descriptions**

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0000	CONTROL	R/W	N	Power-on-Reset : 0x0	Bit 0: SW_ENABLE Bit 1: REG_UPDATE Bit 4: BYPASS(1) Bit 5: TEST_PATTERN(1) Bit 30: FRAME_SYNC_RESET (1: reset) Bit 31: SW_RESET (1: reset)
0x0004	STATUS	R/W	No	0	Bit 0: PROC_STARTED Bit 1: EOF Bit 16: SLAVE_ERROR
0x0008	ERROR	R/W	No	0	Bit 0: SLAVE_EOL_EARLY Bit 1: SLAVE_EOL_LATE Bit 2: SLAVE_SOF_EARLY Bit 3: SLAVE_SOF_LATE
0x000C	IRQ_ENABLE	R/W	No	0	16-0: Interrupt enable bits corresponding to STATUS bits
0x0010	VERSION	R	N/A	0x0400A001	7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR
0x0014	SYSDEBUG0	R	N/A	0	0-31: Frame Throughput monitor(1)
0x0018	SYSDEBUG1	R	N/A	0	0-31: Line Throughput monitor(1)
0x001C	SYSDEBUG2	R	N/A	0	0-31: Pixel Throughput monitor(1)

Table 2-10: **Register Names and Descriptions**

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0020	ACTIVE_SIZE	R/W	Yes	Specified via GUI	12-0: Number of Active Pixels per Scanline 28-16: Number of Active Lines per Frame
0x0100	FILT_STRENGTH	R/W	Yes	Specified via GUI	Strength of the smoothing filter Possible values are 0, 1, 2, 3, and 4 0 = Bypass the smoothing filter 1 = Weakest (less noise reduction) 4 = Strongest (more noise reduction)

1. Only available when the debugging features option is enabled in the GUI at the time the core is instantiated.

CONTROL (0x0000) Register

Bit 0 of the `CONTROL` register, `SW_ENABLE`, facilitates enabling and disabling the core from software. Writing '0' to this bit effectively disables the core halting further operations, which blocks the propagation of all video signals. After Power up, or Global Reset, the `SW_ENABLE` defaults to 0 for the AXI4-Lite interface. Similar to the `ACLKEN` pin, the `SW_ENABLE` flag is not synchronized with the AXI4-Stream interfaces: Enabling or Disabling the core takes effect immediately, irrespective of the core processing status. Disabling the core for extended periods may lead to image tearing.

Bit 1 of the `CONTROL` register, `REG_UPDATE` is a write done semaphore for the host processor, which facilitates committing all user and timing register updates simultaneously. The Image Noise Reduction core `ACTIVE_SIZE` and `FILT_STRENGTH` registers are double buffered. One set of registers (the processor registers) is directly accessed by the processor interface, while the other set (the active set) is actively used by the core. New values written to the processor registers will get copied over to the active set at the end of the AXI4-Stream frame, if and only if `REG_UPDATE` is set. Setting `REG_UPDATE` to 0 before updating multiple register values, then setting `REG_UPDATE` to 1 when updates are completed ensures all registers are updated simultaneously at the frame boundary without causing image tearing.

Bit 4 of the `CONTROL` register, `BYPASS`, switches the core to bypass mode if debug features are enabled. In bypass mode the Image Noise Reduction core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output. Refer to Debugging Features in Appendix C for more information. If debug features were not included at instantiation, this flag has no effect on the operation of the core. Switching bypass mode on or off is not synchronized to frame processing, therefore can lead to image tearing.

Bit 5 of the CONTROL register, TEST_PATTERN, switches the core to test-pattern generator mode if debug features are enabled. Refer to Debugging Features in Appendix C for more information. If debug features were not included at instantiation, this flag has no effect on the operation of the core. Switching test-pattern generator mode on or off is not synchronized to frame processing, therefore can lead to image tearing.

Bits 30 and 31 of the CONTROL register, FRAME_SYNC_RESET and SW_RESET facilitate software reset. Setting SW_RESET reinitializes the core to GUI default values, all internal registers and outputs are cleared and held at initial values until SW_RESET is set to 0. The SW_RESET flag is not synchronized with the AXI4-Stream interfaces. Resetting the core while frame processing is in progress will cause image tearing. For applications where the soft-ware reset functionality is desirable, but image tearing has to be avoided a frame synchronized software reset (FRAME_SYNC_RESET) is available. Setting FRAME_SYNC_RESET to 1 will reset the core at the end of the frame being processed, or immediately if the core is between frames when the FRAME_SYNC_RESET was asserted. After reset, the FRAME_SYNC_RESET bit is automatically cleared, so the core can get ready to process the next frame of video as soon as possible. The default value of both RESET bits is 0. Core instances with no AXI4-Lite control interface can only be reset via the ARESETn pin.

STATUS (0x0004) Register

All bits of the STATUS register can be used to request an interrupt from the host processor. To facilitate identification of the interrupt source, bits of the STATUS register remain set after an event associated with the particular STATUS register bit, even if the event condition is not present at the time the interrupt is serviced.

Bits of the STATUS register can be cleared individually by writing '1' to the bit position to be cleared.

Bit 0 of the STATUS register, PROC_STARTED, indicates that processing of a frame has commenced via the AXI4-Stream interface.

Bit 1 of the STATUS register, End-of-frame (EOF), indicates that the processing of a frame has completed.

Bit 16 of the STATUS register, SLAVE_ERROR, indicates that one of the conditions monitored by the ERROR register has occurred.

ERROR (0x0008) Register

Bit 16 of the STATUS register, SLAVE_ERROR, indicates that one of the conditions monitored by the ERROR register has occurred. This bit can be used to request an interrupt from the host processor. To facilitate identification of the interrupt source, bits of the STATUS and ERROR registers remain set after an event associated with the particular ERROR register bit, even if the event condition is not present at the time the interrupt is serviced.

Bits of the ERROR register can be cleared individually by writing '1' to the bit position to be cleared.

Bit 0 of the ERROR register, EOL_EARLY, indicates an error during processing a video frame via the AXI4-Stream slave port. The number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the ACTIVE_SIZE register.

Bit 1 of the ERROR register, EOL_LATE, indicates an error during processing a video frame via the AXI4-Stream slave port. The number of pixels received between the last EOL signal surpassed the value programmed into the ACTIVE_SIZE register.

Bit 2 of the ERROR register, SOF_EARLY, indicates an error during processing a video frame via the AXI4-Stream slave port. The number of pixels received between the latest and the preceding Start-Of-Frame (SOF) signal was less than the value programmed into the ACTIVE_SIZE register.

Bit 3 of the ERROR register, SOF_LATE, indicates an error during processing a video frame via the AXI4-Stream slave port. The number of pixels received between the last SOF signal surpassed the value programmed into the ACTIVE_SIZE register.

IRQ_ENABLE (0x000C) Register

Any bits of the STATUS register can generate a host-processor interrupt request via the IRQ pin. The Interrupt Enable register facilitates selecting which bits of STATUS register will assert IRQ. Bits of the STATUS registers are masked by (AND) corresponding bits of the IRQ_ENABLE register and the resulting terms are combined (OR) together to generate IRQ.

Version (0x0010) Register

Bit fields of the Version Register facilitate software identification of the exact version of the hardware peripheral incorporated into a system. The core driver can take advantage of this Read-Only value to verify that the software is matched to the correct version of the hardware.

SYSDEBUG0 (0x0014) Register

The SYSDEBUG0, or Frame Throughput Monitor, register indicates the number of frames processed since power-up or the last time the core was reset. The SYSDEBUG registers can be useful to identify external memory / Frame buffer / or throughput bottlenecks in a video system. Refer to Debugging Features in Appendix C for more information.

SYSDEBUG1 (0x0018) Register

The SYSDEBUG1, or Line Throughput Monitor, register indicates the number of lines processed since power-up or the last time the core was reset. The SYSDEBUG registers can be useful to identify external memory / Frame buffer / or throughput bottlenecks in a video system. Refer to Debugging Features in Appendix C for more information.

SYSDEBUG2 (0x001C) Register

The SYSDEBUG2, or Pixel Throughput Monitor, register indicates the number of pixels processed since power-up or the last time the core was reset. The SYSDEBUG registers can be useful to identify external memory / Frame buffer / or throughput bottlenecks in a video system. Refer to Debugging Features in Appendix C for more information.

ACTIVE_SIZE (0x0020) Register

The ACTIVE_SIZE register encodes the number of active pixels per scan line and the number of active scan lines per frame. The lower half-word (bits 12:0) encodes the number of active pixels per scan line. Supported values are between 32 and the value provided in the **Maximum number of pixels per scan line** field in the GUI. The upper half-word (bits 28:16) encodes the number of active lines per frame. Supported values are 32 to 7680. To avoid processing errors, the user should restrict values written to ACTIVE_SIZE to the range supported by the core instance.

FILT_STRENGTH (0x0100) Register

The FILT_STRENGTH register indicates which smoothing filter to use. The possible filter strength values are 0, 1, 2, 3, and 4. A value of 1 is the weakest filter (less noise reduction) and 4 is the strongest (more noise reduction). Setting the filter strength to 0 will bypass the smoothing filter. See Chapter 4, Designing with the Core for details on each filter.

Interrupt Subsystem

STATUS register bits can trigger interrupts so embedded application developers can quickly identify faulty interfaces or incorrectly parameterized cores in a video system. Irrespective of whether the AXI4-Lite control interface is present or not, the Image Noise Reduction core detects AXI4-Stream framing errors, as well as the beginning and the end of frame processing.

When the core is instantiated with an AXI4-Lite Control interface, the optional interrupt request pin (IRQ) is present. Events associated with bits of the STATUS register can generate a (level triggered) interrupt, if the corresponding bits of the interrupt enable register (IRQ_ENABLE) are set. Once set by the corresponding event, bits of the STATUS register stay set until the user application clears them by writing '1' to the desired bit positions. Using this mechanism the system processor can identify and clear the interrupt source.

Without the AXI4-Lite interface the user can still benefit from the core signaling error and status events. By selecting the **Enable INTC Port** check-box on the GUI, the core generates the optional `INTC_IF` port. This vector of signals gives parallel access to the individual interrupt sources, as seen in Table 2-11.

Unlike `STATUS` and `ERROR` flags, `INTC_IF` signals are not held, rather stay asserted only while the corresponding event persists.

Table 2-11: **INTC_IF Signal Functions**

INTC_IF signal	Function
0	Frame processing start
1	Frame processing complete
2	Pixel counter terminal count
3	Line counter terminal count
4	Slave error
5	EOL Early
6	EOL Late
7	SOF Early
8	SOF Late

In a system integration tool, such as EDK, the interrupt controller INTC IP can be used to register the selected `INTC_IF` signals as edge triggered interrupt sources. The INTC IP provides functionality to mask (enable or disable), as well as identify individual interrupt sources from software. Alternatively, for an external processor or MCU the user can custom build a priority interrupt controller to aggregate interrupt requests and identify interrupt sources.

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 or EDK GUIs. 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.

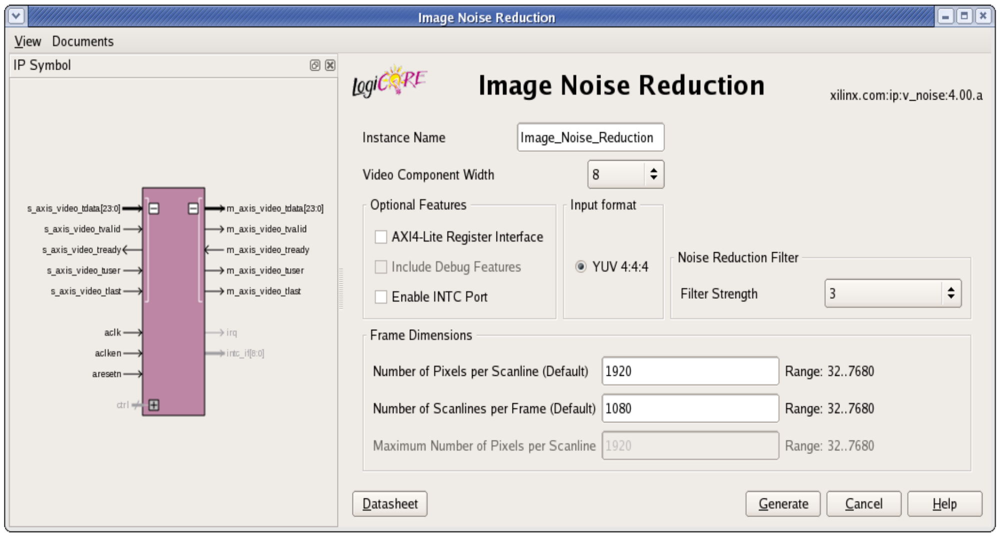


Image Noise Reduction

xilinx.com:ip:v_noise:4.00.a

Instance Name	Image_Noise_Reduction
Video Component Width	8

Optional Features
- ☐ AXI4-Lite Register Interface
- ☐ Include Debug Features
- ☐ Enable INTC Port

Input format
- ● YUV 4:4:4

Noise Reduction Filter

Filter Strength	3

Frame Dimensions

Number of Pixels per Scanline (Default)	1920	Range: 32..7680
Number of Scanlines per Frame (Default)	1080	Range: 32..7680
Maximum Number of Pixels per Scanline	1920	Range: 32..7680

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

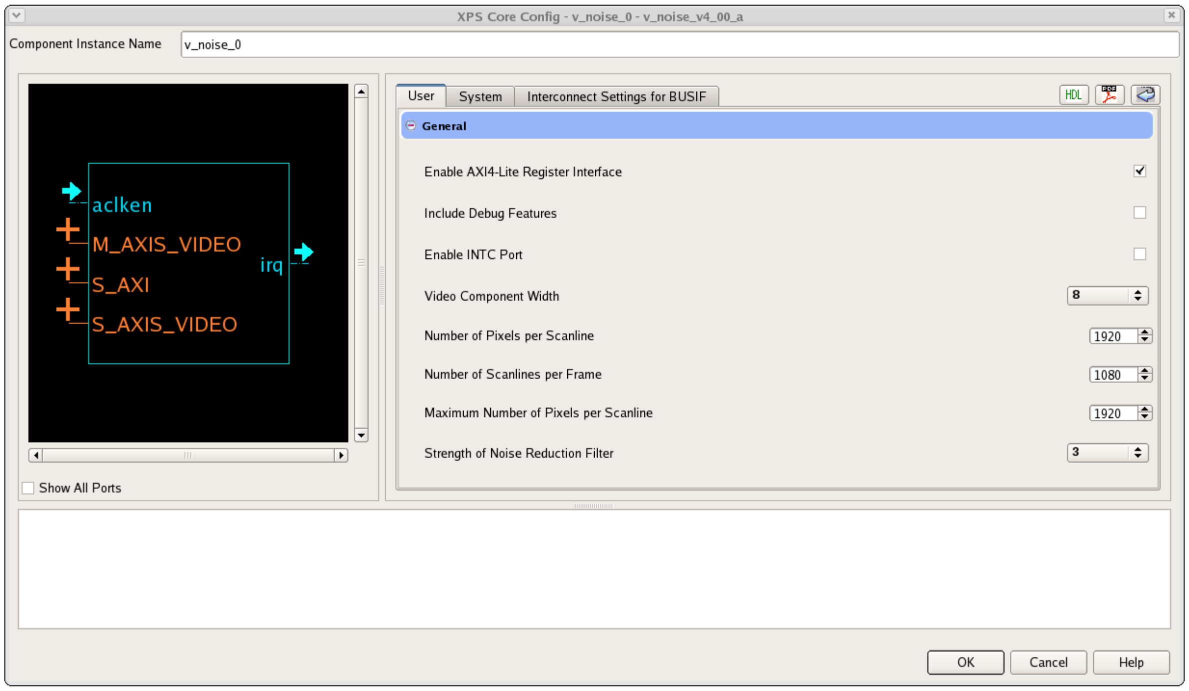
The GUI displays a representation of the IP symbol on the left side, and the parameter assignments on the right side, which are described as follows:

- **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 "_". The name `v_noise_v4_00_a` cannot be used as a component name.
- **Video Component Width:** Specifies the bit width of input samples. Permitted values are 8, 10 and 12 bits.

- **Optional Features**:
    - **AXI4-Lite Register Interface**: When selected, the core will be generated with an AXI4-Lite interface, which gives access to dynamically program and change processing parameters. For more information, refer to Control Interface in Chapter 2.
    - **Include Debugging Features**: When selected, the core will be generated with debugging features, which simplify system design, testing and debugging. For more information, refer to Debugging Features in Appendix C.

      Note: Debugging features are only available when the AXI4-Lite Register Interface is selected.
    - **INTC Interface**: When selected, the core will generate the optional `INTC_IF` port, which gives parallel access to signals indicating frame processing status and error conditions. For more information, refer to Interrupt Subsystem in Chapter 2.
- **Input Frame Dimensions**:
    - **Number of Active Pixels per Scan line**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the CORE Generator GUI as the default value for the lower half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the horizontal size of the frames the generated core instance is to process.
    - **Number of Active Lines per Frame**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the CORE Generator GUI as the default value for the upper half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the vertical size (number of lines) of the frames the generated core instance is to process.
    - **Maximum Number of Active Pixels Per Scan line**: Specifies the maximum number of pixels per scan line that can be processed by the generated core instance. Permitted values are from 32 to 7680. Specifying this value is necessary to establish the depth of internal line buffers. The actual value selected for Number of Active Pixels per Scan line, or the corresponding lower half-word of the `ACTIVE_SIZE` register must always be less than the value provided by Maximum Number of Active Pixels Per Scan line. Using a tight upper-bound results in optimal block RAM usage. This field is enabled only when the AXI4-Lite interface is selected. Otherwise contents of the field are reflecting the actual contents of the **Number of Active Pixels per Scan line** field as for constant mode the maximum number of pixels equals the active number of pixels.
- **Filter Strength:** Specifies which of the four smoothing filters to use. The allowed values are 0, 1, 2, 3, and 4. Filter Strength of 1 provides the weakest smoothing, and Filter Strength of 4 provides the strongest smoothing. Therefore, Filter Strength of 4 provides the most noise reduction. A Filter Strength of zero will bypass the smoothing filter.

Definitions of the EDK GUI controls are identical to the corresponding CORE Generator settings, as shown in Figure 3-2.



Component Instance Name: v_noise_0

Enable AXI4-Lite Register Interface ✓
Include Debug Features
Enable INTC Port
Video Component Width: 8
Number of Pixels per Scanline: 1920
Number of Scanlines per Frame: 1080
Maximum Number of Pixels per Scanline: 1920
Strength of Noise Reduction Filter: 3

Figure 3-2: **EDK GUI Screen**

Parameter Values in the XCO File

Table 3-1 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.

Table 3-1: **XCO Parameters**

XCO parameter	Default	Valid Values
component_name	noise_reduction	ASCII text using characters: a..z, 0..9 and "_" starting with a letter. Note: "v_noise_v4_00_a" is not allowed.
s_axis_video_data_width	8	8, 10, 12
s_axis_video_format	1	1
has_axi4_lite	false	true, false
has_intc_if	false	true, false
has_debug	false	true, false
active_cols	1920	32 – 7680
active_rows	1080	32 - 7680

Table 3-1: **XCO Parameters *(Cont'd)***

XCO parameter	Default	Valid Values
max_cols	1920	32 - 7680
filt_strength	3	0, 1, 2, 3, 4

Output Generation

CORE Generator outputs 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 directory>.

File Details

The CORE Generator output consists of some or all the following files, as shown in Table 3-2.

Table 3-2: **Files Generated by CORE Generator**

Name	Description
<component_name>.xco	CORE Generator input file containing the parameters used to generate a core.
<component_name>.ngc	Binary Xilinx implementation netlist files containing the information required to implement the module in a Xilinx FPGA.
<component_name>.vho <component_name>.veo	Template files containing code that can be used as a model for instantiating.
<component_name>.vhd <component_name>.v	Structural simulation model.
/doc/pg011_v_noise.pdf /doc/v_noise_v4_00_a_vinfo.html	Core documents.
<component_name>.asy	Graphical symbol information file. Used by the ISE tools and some third party tools to create a symbol representing the core.
<component_name>_xmdf.tcl	ISE Project Navigator interface file. ISE uses this file to determine how the files output by CORE Generator for the core can be integrated into your ISE project.
<component_name>.gise <component_name>.xise	ISE Project Navigator support files. These are generated files and should not be edited directly.
<component_name>_readme.txt	Readme file for the IP.
<component_name>_flist.txt	Text file listing all of the output files produced when a customized core was generated in the CORE Generator.

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.

$$\text{Filter 1} = \begin{bmatrix} 121 \\ 161 \\ 121 \end{bmatrix}$$

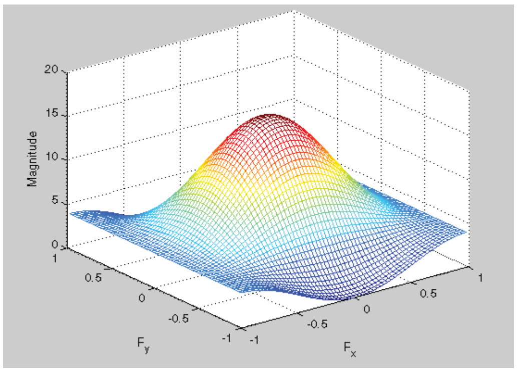


Figure 4-1: **Coefficients and Frequency Response for Filter Strength 1**

$$\text{Filter 2} = \begin{bmatrix}121\\242\\121\end{bmatrix}$$

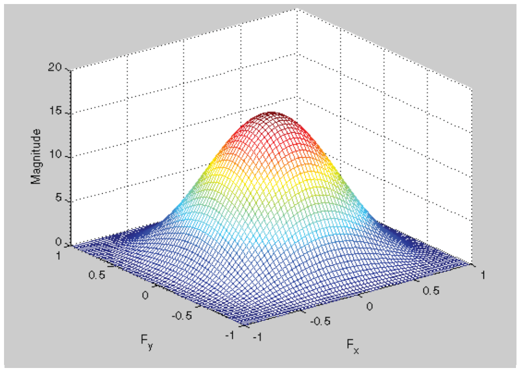


Figure 4-2: **Coefficients and Frequency Response for Filter Strength 2**

$$\text{Filter 3} = \begin{bmatrix}01110\\12421\\01110\end{bmatrix}$$

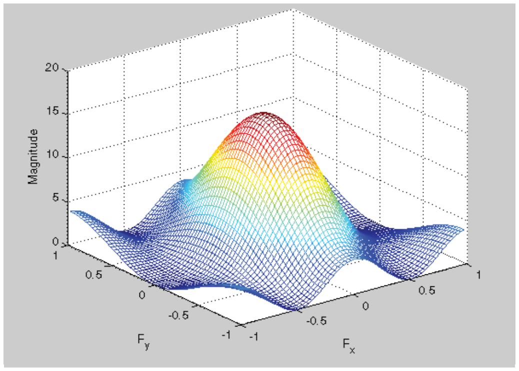


Figure 4-3: **Coefficients and Frequency Response for Filter Strength 3**

$$\text{Filter 4} = \begin{bmatrix}01210\\12221\\01210\end{bmatrix}$$

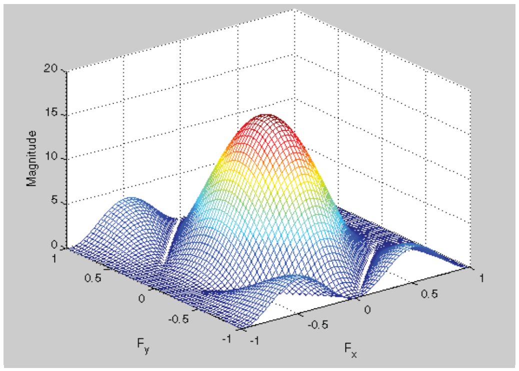


Figure 4-4: **Coefficients and Frequency Response for Filter Strength 4**

The Image Noise Reduction core processes samples provided via an AXI4-Stream Video Protocol slave interface, outputs pixels via an AXI4-Stream Video Protocol master interface, and can be controlled via an optional AXI4-Lite interface. The Image Noise Reduction block cannot change the input/output image sizes, the input and output pixel clock rates, or the frame rate. It is recommended that the Image Noise Reduction core is used in conjunction with the Video In to AXI4-Stream and Video Timing Controller cores. The Video Timing Controller core measures the timing parameters, such as number of active scan lines, number of active pixels per scan line of the input video stream. The Video In to AXI4-Stream core converts the incoming video stream to AXI4-Stream Video Protocol.

Typically, the Image Noise Reduction core is part of a larger system such as the Image Sensor Pipeline (ISP) System, as shown in Figure 4-5.

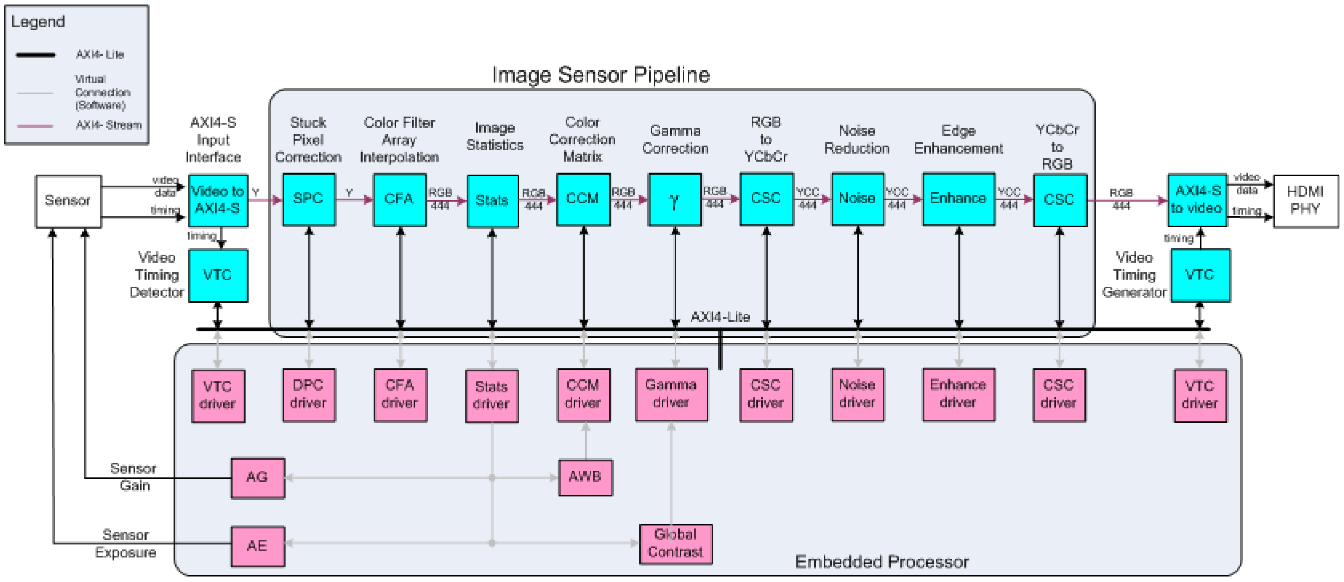


Figure 4-5: **Image Sensor Pipeline System with Image Noise Reduction Core**

Clock, Enable, and Reset Considerations

ACLK

The master and slave AXI4-Stream video interfaces use the ACLK clock signal as their shared clock reference, as shown in Figure 4-6.

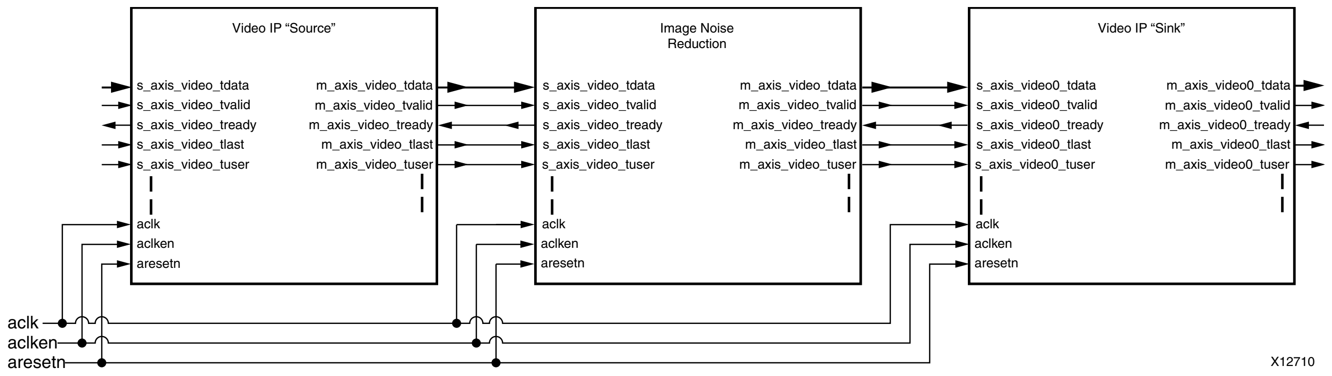


Figure 4-6: **Example of ACLK Routing in an ISP Processing Pipeline**

The ACLK pin is also shared between the AXI4-Lite and AXI4-Stream interfaces, the Image Noise Reduction core does not contain optional clock-domain crossing logic. If in the user system the AXI4-Lite Control interface clock (CLK_LITE) is different from the AXI4-Stream clock (CLK_STREAM), and

- ($F_{CLK\_STREAM} > F_{CLK\_LITE}$) then clock-domain crossing logic needs to be inserted in front of the AXI4-Lite Control interface and the Image Noise Reduction core can be clocked at the AXI4-Stream clock via ACLK,
- ($F_{CLK\_STREAM} < F_{CLK\_LITE}$) then clock-domain crossing logic needs to be inserted before the AXI4-Stream interface, and the Image Noise Reduction core needs to be clocked at the AXI4-Lite clock via the ACLK pin, as shown in Figure 4-7. Alternatively, if F_{CLK_LITE} greater than of the F_{MAX} of the Image Noise Reduction core, clock domain crossing logic can be inserted in front of the AXI4-Lite Control interface.

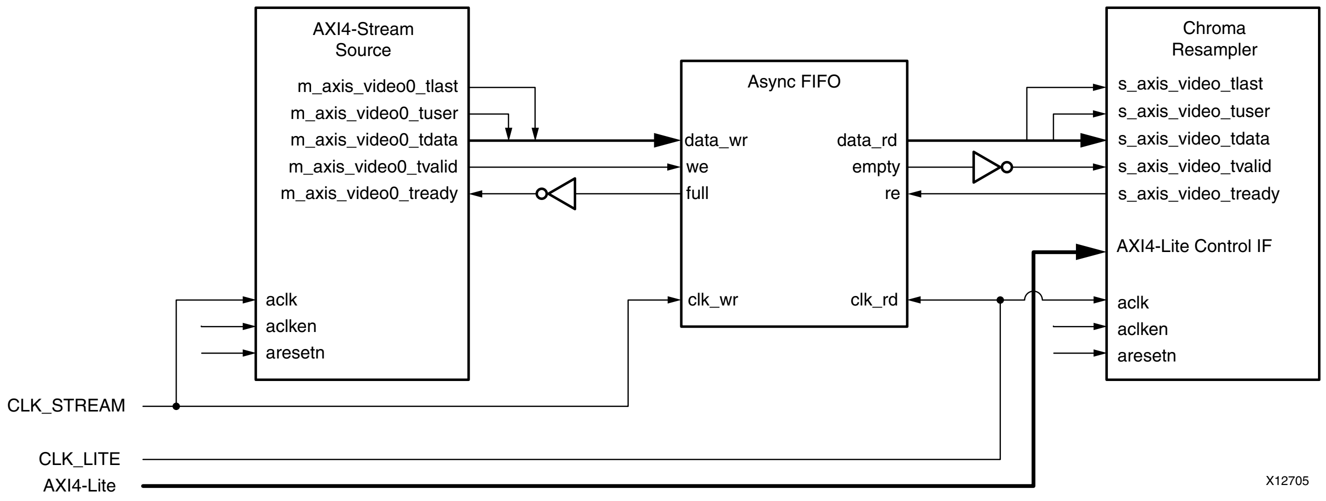


Figure 4-7: **Image Noise Reduction Core Top-Level Signaling Interface**

In either case, Xilinx System Integrator tools, such as EDK, can automatically infer clock-domain crossing logic using the AXI interconnect core, when the tool detects that the master / slave side of AXI4 interfaces operate on different CLK rates. For manual instantiation of clock-domain crossing logic, HDL users can take advantage of the FIFO Generator IP core, as shown in Figure 4-7.

ACLKEN

The Image Noise Reduction core has two enable options: the ACLKEN pin (hardware clock enable), and the software reset option provided via the AXI4-Lite control interface (when present).

ACLKEN is by no means synchronized internally to AXI4-Stream frame processing therefore de-asserting ACLKEN for extended periods of time may lead to image tearing.

The ACLKEN pin facilitates:

- Multi-cycle path designs (high speed clock division without clock gating)
- Standby operation of subsystems to save on power
- Hardware controlled bring-up of system components

Note: When ACLKEN (clock enable) pins are used (toggled) in conjunction with a common clock source driving the master and slave sides of an AXI4-Stream interface, to prevent transaction errors the ACLKEN pins associated with the master and slave component interfaces must also be driven by the same signal (Figure 4-6).

Note: When two cores connected via AXI4-Stream interfaces, where only the master or the slave interface has an ACLKEN port, which is not permanently tied high, the two interfaces should be connected via the AXI4-Stream Interconnect or AXI-FIFO cores to avoid data corruption (Figure 4-7).

ARESETn

The Image Noise Reduction core has two reset source: the ARESETn pin (hardware reset), and the software reset option provided via the AXI4-Lite control interface (when present).

Note: ARESETn is by no means synchronized internally to AXI4-Stream frame processing, therefore de-asserting ARESETn while a frame is being process will lead to image tearing.

The external reset pulse needs to be held for 32 ACLK cycles to reset the core.

Note: When a system with multiple-clocks and corresponding reset signals are being reset, the reset generator has to ensure all reset signals are asserted/de-asserted long enough that all interfaces and clock-domains in all IP cores are correctly reinitialized.

System Considerations

When using the Image Noise Reduction, it needs to be configured for the actual image frame size, to operate properly. To gather the frame size information from the incoming video stream, it can be connected to the Video In to AXI4-Stream input and the Video Timing Controller. The timing detector logic in the Video Timing Controller will gather the image sensor timing signals. The AXI4-Lite control interface on the Video Timing Controller allows the system processor to read out the measured frame dimensions, and program all downstream cores, such as the Image Noise Reduction, with the appropriate image dimensions.

If the target system uses a static incoming video resolution and does not need to adjust the noise reduction parameters of this core, the user can choose to consolidate the active-size and filter strength values, and create a constant configuration by removing the AXI4-Lite interface. This option allows reducing the core Slice footprint.

Programming Sequence

If processing parameters such as the image size needs to be changed on the fly, or the system needs to be reinitialized, it is recommended that pipelined Xilinx IP video cores are disabled/reset from system output towards the system input, and programmed/enabled from system input to system output. STATUS register bits allow system processors to identify the processing states of individual constituent cores, and successively disable a pipeline as one core after another is finished processing the last frame of data.

Error Propagation and Recovery

Parameterization and/or configuration registers define the dimensions of video frames video IP should process. Starting from a known state, based on these configuration settings the IP can predict when the beginning of the next frame is expected. Similarly, the IP can predict when the last pixel of each scan line is expected. SOF detected before it was expected (early), or SOF not present when it is expected (late), EOL detected before expected (early), or EOL not present when expected (late), signals error conditions indicative of either upstream communication errors or incorrect core configuration.

When SOF is detected early, the output SOF signal is generated early, terminating the previous frame immediately. When SOF is detected late, the output SOF signal is generated according to the programmed values. Extra lines / pixels from the previous frame are dropped until the input SOF is captured.

Similarly, when EOL is detected early, the output EOL signal is generated early, terminating the previous line immediately. When EOL is detected late, the output EOL signal is generated according to the programmed values. Extra pixels from the previous line are dropped until the input EOL is captured.

Constraining the Core

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

Required Constraints

The ACLK pin should be constrained at the pixel clock rate desired for your video stream.

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for the Image Noise Reduction core. For a complete listing of supported devices, see the release notes for this core.

Clock Frequencies

The pixel clock frequency is the required frequency for the Image Noise Reduction core. See Maximum Frequencies in Chapter 2.

Clock Management

There is only one clock for the Image Noise Reduction core.

Clock Placement

There are no specific Clock placement requirements for the Image Noise Reduction core.

Banking

There are no specific Banking rules for the Image Noise Reduction core.

Transceiver Placement

There are no Transceiver Placement requirements for the Image Noise Reduction core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for the Image Noise Reduction core.

Detailed Example Design

No example design is available at this time. For a comprehensive listing of Video and Imaging application notes, white papers, related IP cores including the most recent reference designs available, see the Video and Imaging Resources page at www.xilinx.com/esp/video/refdes_listing.htm.

Demonstration Test Bench

A demonstration test bench is provided which enables core users to observe core behavior in a typical use scenario. The user is encouraged to make simple modifications to the test conditions and observe the changes in the waveform.

Test Bench Structure

The top-level entity, `tb_main.v`, instantiates the following modules:

- DUT

  The Image Noise Reduction v4.00.a core instance under test.

- axi4lite_mst

  The AXI4-Lite master module, which initiates AXI4-Lite transactions to program core registers.

- axi4s_video_mst

  The AXI4-Stream master module, which opens the stimuli TXT file and initiates AXI4-Stream transactions to provide stimuli data for the core

- axi4s_video_slv

  The AXI4-Stream slave module, which opens the result TXT file and verifies AXI4-Stream transactions from the core

- ce_gen

  Programmable Clock Enable (ACLKEN) generator

Running the Simulation

- Simulation using ModelSim for Linux:
  From the console, Type "source run_mti.sh".
- Simulation using iSim for Linux:
  From the console, Type "source run_isim.sh".
- Simulation using ModelSim for Windows:
  Double-click on "run_mti.bat" file.
- Simulation using iSim:
  Double-click on "run_isim.bat" file.

Directory and File Contents

The directory structure underneath the top-level folder is:

- 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 simulation files and the XCO CORE Generator parameterization file of the core instance. The VHD file is a netlist generated using CORE Generator. The XCO file can be used to regenerate a new netlist using CORE Generator.

The available core C-model can be used to generate stimuli and expected results for any user image. For more information, refer to Appendix E, Using the C Model.

The top-level directory contains packages and Verilog modules used by the test bench, as well as:

- isim_wave.wcfg:
  Waveform configuration for ISIM

- mti_wave.do:
  Waveform configuration for ModelSim
- run_isim.bat :
  Runscript for iSim in Windows
- run_isim.sh:
  Runscript for iSim in Linux
- run_mti.bat:
  Runscript for ModelSim in Windows
- run_mti.sh:
  Runscript for ModelSim in Linux

Verification, Compliance, and Interoperability

This chapter contains details about verification for the Image Noise Reduction core.

Simulation

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

- Register accesses
- Processing multiple frames of data
- AXI4-Stream bidirectional data-throttling tests
- Testing detection, and recovery from various AXI4-Stream framing error scenarios
- Testing different ACLKEN and ARESETn assertion scenarios
- Testing of various frame sizes
- Varying parameter settings

Hardware Testing

The Image Noise Reduction core has been validated in hardware at Xilinx to represent a variety of parameterizations, including the following:

- A test design was developed for the core that incorporated a MicroBlaze™ processor, AXI4-Lite interconnect and various other peripherals. The software for the test system included pre-generated input and output data along with live video stream. The MicroBlaze processor was responsible for:
    - Initializing the appropriate input and output buffers
    - Initializing the Image Noise Reduction core
    - Launching the test

- Comparing the output of the core against the expected results
- Reporting the Pass/Fail status of the test and any errors that were found

Interoperability

The core slave (input) AXI4-Stream interface can work directly with the Video In to AXI4-Stream core. The core master (output) YCbCR 4:4:4 interface can work directly with any Video core that consumes YCbCr 4:4:4 data. The AXI4-Stream interfaces must be compliant with the AXI4-Stream Video Protocol as described in "Video IP: AXI Feature Adoption" of UG761, *Xilinx AXI Reference Guide*.

Migrating

From version v3.0 to v4.00.a of the Image Noise Reduction core the following significant changes took place:

- XSVI interfaces were replaced by AXI4-Stream interfaces.
- Since AXI4-Stream does not carry video timing data, the timing detector and timing generator modules were trimmed.
- The pCore, General Purpose Processor and Constant modes became obsolete and were removed.
- Native support for EDK have been added - the Image Noise Reduction core appears in the EDK IP Catalog.
- Debugging features have been added.
- The AXI4-Lite control interface register map is standardized between Xilinx video cores.

Because of the complex nature of these changes, replacing a v3.0 version of the core in a customer design is not trivial. An existing EDK pCore, or Constant Image Noise Reduction instance can be converted from XSVI to AXI4-Stream, using the Video In to AXI4-Stream core or components from XAPP521 (v1.0), *Bridging Xilinx Streaming Video Interface with the AXI4-Stream Protocol*.

A v3.0 pCore instance in EDK can be replaced from v4.00.a directly from the EDK IP Catalog. However, the application software needs to be updated for the changed functionality and addresses of the `IRQ_ENABLE`, `STATUS`, `ERROR`, and `FILT_STRENGTH` registers. Consider replacing a legacy Image Noise Reduction pCore from EDK with a v4.00.a instance without AXI4-Lite interface to save resources.

If an ISE design uses the General Purpose Processor interface, the following steps may be necessary:

- Timing detection, generation using the Video Timing Controller core.
- Replacing XSVI interfaces with conversion modules described in XAPP521 or trying the Video In to AXI4-Stream core.
- Updating the Image Noise Reduction core instance to v4.00.a with or without AXI4-Lite interface.
- The INTC interface and debug functionality are new features for v4.00.a. When migrating an existing design, these functions may be disabled.

Debugging

It is recommended to prototype the system with the AXI4-Stream interface enabled, so status and error detection, reset, and dynamic size programming can be used during debugging.

The following steps are recommended to bring-up/debug the core in a video/imaging system:

1. Bring up the AXI4-Lite interface
2. Bring up the AXI4-Stream interfaces
    - (Optional) Balancing throughput

Once the core is working as expected, the user may consider 'hardening' the configuration by replacing the Image Noise Reduction core with an instance where GUI default values are set to the established `ACTIVE_SIZE` and `FILT_STRENGTH` values, but the AXI4-Lite interface is disabled. This configuration reduces the core slice footprint.

Bringing up the AXI4-Lite Interface

Table C-1 describes how to troubleshoot the AXI4-Lite interface.

Table C-1: **Troubleshooting the AXI4-Lite Interface**

Symptom	Solution
Readback value for the VERSION_REGISTER is different from expected default values	Does the core receive ACLK? Is the core enabled? Set ACLKEN=1 Is the core in reset? Set ARESETn=1. The address maps between software and hardware could get out of sync. Regenerate addresses in EDK, make sure the MHS file in EDK and the xparameters.h in the SDK project are up to date.

Table C-1: **Troubleshooting the AXI4-Lite Interface**

Symptom	Solution
Readback values from values are stuck, and cannot be overwritten.	Is the target address writable?
The interface is unreliable. Subsequent reads from the same address return different value, readback values differ from values written to the same address.	Clock domain crossing issues between the host processor and the peripheral. HDL users need to make sure the AXI4-Lite Master is in the same clock domain as the AXI4-Lite Slave. If not, proper clock-domain crossing logic, such as asynchronous FIFOs need to be inserted.

Assuming the AXI4-Lite interface works, the second step is to bring up the AXI4-Stream interfaces.

Bringing up the AXI4-Stream Interfaces

Table C-2 describes how to troubleshoot the AXI4-Stream interface.

Table C-2: **Troubleshooting AXI4-Stream Interface**

Symptom	Solution
Bit 0 of the `ERROR` register reads back set.	Bit 0 of the `ERROR` register, EOL_EARLY, indicates the number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the `ACTIVE_SIZE` register. If the value was provided by the Video Timing Controller core, read out `ACTIVE_SIZE` register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using the ChipScope tool, measure the number of active AXI4-Stream transactions between EOL pulses.
Bit 1 of the `ERROR` register reads back set.	Bit 1 of the `ERROR` register, EOL_LATE, indicates the number of pixels received between the last End-Of-Line (EOL) signal surpassed the value programmed into the `ACTIVE_SIZE` register. If the value was provided by the Video Timing Controller core, read out `ACTIVE_SIZE` register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using the ChipScope tool, measure the number of active AXI4-Stream transactions between EOL pulses.
Bit 2 or Bit 3 of the `ERROR` register reads back set.	Bit 2 of the `ERROR` register, SOF_EARLY, and bit 3 of the `ERROR` register SOF_LATE indicate the number of pixels received between the latest and the preceding Start-Of-Frame (SOF) differ from the value programmed into the `ACTIVE_SIZE` register. If the value was provided by the Video Timing Controller core, read out `ACTIVE_SIZE` register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using the ChipScope tool, measure the number EOL pulses between subsequent SOF pulses.
s_axis_video_tready stuck low, the upstream core cannot send data.	During initialization, line-, and frame-flushing, the Image Noise Reduction core keeps its `s_axis_video_tready` input low. Afterwards, the core should assert `s_axis_video_tready` automatically. Is `m_axis_video_tready` low? If so, the Image Noise Reduction core cannot send data downstream, and the internal FIFOs are full.

Table C-2: **Troubleshooting AXI4-Stream Interface**

Symptom	Solution
m_axis_video_tvalid stuck low, the downstream core is not receiving data	1. No data is generated during the first two lines of processing. 2. If the programmed active number of pixels per line is radically smaller than the actual line length, the core drops most of the pixels waiting for the (`s_axis_video_tlast`) End-of-line signal. Check the ERROR register.
Generated SOF signal (m_axis_video_tuser0) signal misplaced.	Check the ERROR register.
Generated EOL signal (`m_axis_video_tlast`) signal misplaced.	Check the ERROR register.
Data samples lost between Upstream core and the Image Noise Reduction core. Inconsistent EOL and/ or SOF periods received.	1. Are the Master and Slave AXi4-Stream interfaces in the same clock domain? 2. Is proper clock-domain crossing logic instantiated between the upstream core and the Image Noise Reduction core (Asynchronous FIFO)? 3. Did the design meet timing? 4. Is the frequency of the clock source driving the Image Noise Reduction ACLK pin lower than the reported Fmax reached?
Data samples lost between Downstream core and the Image Noise Reduction core. Inconsistent EOL and/ or SOF periods received.	1. Are the Master and Slave AXi4-Stream interfaces in the same clock domain? 2. Is proper clock-domain crossing logic instantiated between the upstream core and the Image Noise Reduction core (Asynchronous FIFO)? 3. Did the design meet timing? 4. Is the frequency of the clock source driving the Image Noise Reduction ACLK pin lower than the reported Fmax reached?

If the AXI4-Stream communication is healthy, but the data seems corrupted, the next step is to find the correct configuration for the Image Noise Reduction core.

Debugging Features

The Image Noise Reduction core is equipped with optional debugging features which aim to accelerate system bring-up, optimize memory and data-path architecture and reduce time to market. The optional debug features can be turned on/off via the **Include Debug Features** checkbox on the GUI when an AXI4-Lite interface is present. Turning off debug features reduces the core Slice footprint.

Core Bypass Option

The bypass option facilitates establishing a straight through connection between input (AXI4-Stream slave) and output (AXI4-Stream master) interfaces bypassing any processing functionality.

Flag BYPASS (bit 4 of the CONTROL register) can turn bypass on (1) or off, when the core instance Debugging Features were enabled at generation. Within the IP this switch controls multiplexers in the AXI4-Stream path.

In bypass mode the Image Noise Reduction core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output.

Starting a system with all processing cores set to bypass, then by turning bypass off from the system input towards the system output allows verification of subsequent cores with known good stimuli.

Built in Test-Pattern Generator

The optional built-in test-pattern generator facilitates to temporarily feed the output AXI4-Stream master interface with a predefined pattern.

Flag TEST_PATTERN (bit 5 of the CONTROL register) can turn test-pattern generation on (1) or off, when the core instance Debugging Features were enabled at generation. Within the IP this switch controls multiplexers in the AXI4-Stream path, switching between the regular core processing output and the test-pattern generator. When enabled, a set of counters generate 256 scan-lines of color-bars, each color bar 64 pixels wide, repetitively cycling through Black, Red, Green, Yellow, Blue, Magenta, Cyan, and White colors till the end of each scan-line. After the Color-Bars segment, the rest of the frame is filled with a monochrome horizontal and vertical ramp.

Starting a system with all processing cores set to test-pattern mode, then by turning test-pattern generation off from the system output towards the system input allows successive bring-up and parameterization of subsequent cores.

Throughput Monitors

Throughput monitors enable the user to monitor processing performance within the core. This information can be used to help debug frame-buffer bandwidth limitation issues, and if possible, allow video application software to balance memory pathways.

Often times video systems, with multi-port access to a shared external memory, have different processing islands. For example a pre-processing sub-system working in the input video clock domain may clean up, transform, and write a video stream, or multiple video streams, to memory. The processing sub-system may read the frames out, process, scale, encode, then write frames back to the frame buffer, in a separate processing clock domain.

Finally, the output sub-system may format the data and read out frames locked to an external clock.

Typically, access to external memory using a multi-port memory controller involves arbitration between competing streams. However, to maximize the throughput of the system, different memory ports may need different specific priorities. To fine tune the arbitration and dynamically balance frame rates, it is beneficial to have access to throughput information measured in different video data paths.

The SYSDEBUG0 (0x0014), or Frame Throughput Monitor, register indicates the number of frames processed since power-up or the last time the core was reset. The SYSDEBUG1 (0x0018), or Line Throughput Monitor, register indicates the number of lines processed since power-up or the last time the core was reset. The SYSDEBUG2 (0x001C), or Pixel Throughput Monitor, register indicates the number of pixels processed since power-up or the last time the core was reset.

Priorities of memory access points can be modified by the application software dynamically to equalize frame, or partial frame rates.

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.

Application Software Development

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

Programmer's Guide

The software API is provided to allow easy access to the Image Noise Reduction AXI4-Lite registers defined in Table 2-10. To utilize the API functions, 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 `v_noise_v4_00_a/examples` subfolder. This file is a sample C program that demonstrates how to use the Image Noise Reduction pCore API.

Table D-1: **Image Noise Reduction Driver Function Definitions**

Function name and parameterization	Description
NOISE_Enable (uint32 BaseAddress)	Enables a Image Noise Reduction instance.
NOISE_Disable (uint32 BaseAddress)	Disables a Image Noise Reduction instance.
NOISE_Reset (uint32 BaseAddress)	Immediately resets a Image Noise Reduction instance. The core stays in reset until the RESET flag is cleared.
NOISE_ClearReset (uint32 BaseAddress)	Clears the reset flag of the core, which allows it to re-sync with the input video stream and return to normal operation.
NOISE_FSync_Reset (uint32 BaseAddress)	Resets a Image Noise Reduction instance at the end of the current frame being processed, or immediately if the core is not currently processing a frame.

Table D-1: **Image Noise Reduction Driver Function Definitions**

Function name and parameterization	Description
NOISE_ReadReg (uint32 BaseAddress, uint32 RegOffset)	Returns the 32-bit unsigned integer value of the register. Read the register selected by RegOffset (defined in Table 2-10).
NOISE_WriteReg (uint32 BaseAddress, uint32 RegOffset, uint32 Data)	Write the register selected by RegOffset (defined in Table 2-10). Data is the 32-bit value to write to the register.
NOISE_RegUpdateEnable (uint32 BaseAddress)	Enables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.
NOISE_RegUpdateDisable (uint32 BaseAddress)	Disables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.

Software Reset

Software reset reinitializes registers of the AXI4-Lite control interface to their initial value, resets FIFOs, forces `m_axis_video_tvalid` and `s_axis_video_tready` to 0. `NOISE_Reset()` and `NOISE_FSync_Reset()` reset the core immediately if the core is not currently processing a frame. If the core is currently processing a frame calling `NOISE_Reset()`, or setting bit 30 of the `CONTROL` register to 1 will cause image tearing. After calling `NOISE_Reset()`, the core remains in reset until `NOISE_ClearReset()` is called.

Calling `NOISE_FSync_Reset()` automates this reset process by waiting until the core finishes processing the current frame, then asserting the reset signal internally, keeping the core in reset only for 32 `ACLK` cycles, then deasserting the signal automatically. After calling `NOISE_FSync_Reset()`, it is not necessary to call `NOISE_ClearReset()` for the core to return to normal operating mode.

Note: Calling `NOISE_FSync_Reset()` does not guarantee prompt, or real-time reseting of the core. If the AXI4-Stream communication is halted mid frame, the core will not reset until the upstream core finishes sending the current frame or starts a new frame.

Double Buffering

Registers `FILT_STRENGTH` and `ACTIVE_SIZE` are double-buffered to ensure no image tearing happens if values are modified during frame processing. Values from the AXI4-Lite interface are latched into processor registers immediately after writing, and processor register values are copied into the active register set at the Start Of Frame (SOF) signal. Double-buffering decouples AXI4-Lite register updates from the AXI4-Stream processing, allowing software a large window of opportunity to update processing parameter values without image tearing.

If multiple register values are changed during frame processing, simple double buffering would not guarantee that all register updates would take effect at the beginning of the same frame. Using a semaphore mechanism, the `RegUpdateEnable()` and `RegUpdateDisable()` functions allows synchronous commitment of register changes. The Image Noise Reduction core will start using the updated `ACTIVE_SIZE` and `FILT_STRENGTH` values only if the `REGUPDATE` flag of the `CONTROL` register is set (1), after the next Start-Of-Frame signal (`s_axis_video_tuser`) is received. Therefore, it is recommended to disable the register update before writing multiple double-buffered registers, then enable register update when register writes are completed.

Reading and Writing Registers

Each software register that is defined in Table 2-10 has a constant that is defined in `noise.h` which is set to the offset for that register listed in Table D-2. It is recommended that the application software uses the predefined register names instead of register values when accessing core registers, so future updates to the Image Noise Reduction drivers which may change register locations will not affect the application dependent on the Image Noise Reduction driver.

Table D-2: **Predefined Constants Defined in noise.h**

Constant Name Definition	Value	Target Register
NOISE_CONTROL	0x0000	CONTROL
NOISE_STATUS	0x0004	STATUS
NOISE_ERROR	0x0008	ERROR
NOISE_IRQ_ENABLE	0x000C	IRQ_ENABLE
NOISE_VERSION	0x0010	VERSION
NOISE_SYSDEBUG0	0x0014	SYSDEBUG0
NOISE_SYSDEBUG1	0x0018	SYSDEBUG1
NOISE_SYSDEBUG2	0x001C	SYSDEBUG2
NOISE_ACTIVE_SIZE	0x0020	ACTIVE_SIZE
NOISE_FILT_STRENGTH	0x0100	FILT_STRENGTH

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 YUV and BIN

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 Using the C Model, page 51. 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_v4_00_a_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_v4_00_a_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
parsers.c	Code for reading configuration file
/examples	Example input files used by C model
noise.cfg	Sample configuration file containing the core parameter settings
input_image.yuv	Sample test image
input_image.hdr	Sample test image header file
/lin	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms
libIp_v_noise_v4_00_a_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_noise_v4_00_a_bitacc_cmodel.so
/lin64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms
libIp_v_noise_v4_00_a_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_noise_v4_00_a_bitacc_cmodel.so
/nt	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms.

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

libIp_v_noise_v4_00_a_bitacc_cmodel.dll lib_Ip_v_noise_v4_00_a_bitacc_cmodel.lib stlport.5.1.dll	Precompiled library files for win32 compilation
/nt64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms.
libIp_v_noise_v4_00_a_bitacc_cmodel.dll lib_Ip_v_noise_v4_00_a_bitacc_cmodel.lib stlport.5.1.dll	Precompiled library files 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_v4_00_a_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_v4_00_a_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_v4_00_a_generics noise_generics;
struct  xilinx_ip_v_noise_v4_00_a_inputs   noise_inputs;
struct  xilinx_ip_v_noise_v4_00_a_outputs  noise_outputs;
```

The declaration of these structures is in the `v_noise_v4_00_a_bitacc_cmodel.h` file.

Table E-3 lists the generic parameters taken by the Image Noise Reduction v4.00.a 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_v4_00_a_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 AXI4-Lite 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_v4_00_a_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. C Model Example Code, page 56 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_v4_00_a_bitacc_simulate(
struct xilinx_ip_v_noise_v4_00_a_generics* generics,
struct xilinx_ip_v_noise_v4_00_a_inputs*   inputs,
struct xilinx_ip_v_noise_v4_00_a_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_v4_00_a_destroy(
struct xilinx_ip_v_noise_v4_00_a_inputs *input,
struct xilinx_ip_v_noise_v4_00_a_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 v4.00.a 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

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

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 file_dir config_file
file_dir : path to the location of the input/output files
config_file: path/name of the configuration file
```

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

This section details the steps to compile the C model with the 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_v4_00_a_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 ../gen_stim.c ../parsers.c -o
   run_bitacc_cmodel -L. -lIp_v_noise_v4_00_a_bitacc_cmodel -Wl,-rpath,.
   ```

```
gcc –m64 -x c++ ../run_bitacc_cmodel.c ../gen_stim.c ../parsers.c -o
run_bitacc_cmodel -L. -lIp_v_noise_v4_00_a_bitacc_cmodel -Wl,-rpath,.
```

Windows (32-bit and 64-bit)

The precompiled library `v_noise_v4_00_a_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_v4_00_a_bitacc_cmodel.lib` to the Resource Files folder of the project
   b. `run_bitacc_cmodel.c`, `gen_stim.c`, and `parsers.c` to the Source Files folder of the project
   c. `v_noise_v4_00_a_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.

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.

For a comprehensive listing of Video and Imaging application notes, white papers, reference designs and related IP cores, see the Video and Imaging Resources page at:

http://www.xilinx.com/esp/video/refdes_listing.htm#ref_des.

References

This document provides supplemental material useful with this product guide:

- UG761, *Xilinx 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 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.
04/24/2012	2.0	Updated core to v4.00.a and ISE tools to v14.1. Replaced XSVI interfaces with AXI4-Stream interfaces. Added native support for EDK.

Notice of Disclaimer

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available “AS IS” and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.

© Copyright 2011-2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.