Image Enhancement
v8.0

LogiCORE IP Product Guide

PG003 November 18, 2015
# Table of Contents

**IP Facts**

---

**Chapter 1: Overview**

Overview ........................................................................................................... 5  
Feature Summary ............................................................................................... 5  
Applications ....................................................................................................... 6  
Licensing and Ordering Information ................................................................. 6  

---

**Chapter 2: Product Specification**

Standards ............................................................................................................ 7  
Performance ........................................................................................................ 7  
Resource Utilization ............................................................................................ 9  
Core Interfaces and Register Space .................................................................. 11  

---

**Chapter 3: Designing with the Core**

General Design Guidelines ................................................................................ 32  
Clock, Enable, and Reset Considerations .......................................................... 33  
System Considerations ....................................................................................... 35  

---

**Chapter 4: Customizing and Generating the Core**

Vivado Integrated Design Environment (IDE) .................................................. 37  
Interface ............................................................................................................ 37  
Output Generation .............................................................................................. 40  

---

**Chapter 5: Constraining the Core**

Required Constraints .......................................................................................... 41  

---

**Chapter 6: Simulation**

---

**Chapter 7: Synthesis and Implementation**

---

**Chapter 8: C Model Reference**

Features .............................................................................................................. 44  

---
Chapter 9: Detailed Example Design

Chapter 10: Test Bench
Demonstration Test Bench .................................................. 55

Appendix A: Verification, Compliance, and Interoperability
Simulation ................................................................. 57
Hardware Testing ......................................................... 57
Interoperability ............................................................ 58

Appendix B: Migrating and Upgrading
Migrating to the Vivado Design Suite .................................. 59
Upgrading in Vivado Design Suite ...................................... 59

Appendix C: Debugging
Finding Help on Xilinx.com ............................................... 62
Debug Tools ............................................................... 63
Hardware Debug .......................................................... 64
Interface Debug .......................................................... 66

Appendix D: Additional Resources
Xilinx Resources .......................................................... 70
References ................................................................. 70
Revision History .......................................................... 71
Notice of Disclaimer ....................................................... 71
Introduction

The Xilinx Image Enhancement LogiCORE™ IP provides users with an easy-to-use IP block to reduce image noise and enhance the edges of objects in each picture. Two-dimensional filters are used to suppress noise while preserving and enhancing edges in the picture.

Features

- Performs image noise reduction
- Performs image edge enhancement
- Optional anti-halo and anti-aliasing for edge enhancement post-processing
- Support for two architectures:
  - High Performance
  - Minimal Resources
- YCbCr 4:4:4 and 4:2:2 input and output
- AXI4-Stream data interfaces
- Optional AXI4-Lite control interface
- Supports 8, 10, 12, and 16 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 at the 24 Hz in supported high performance devices

1. Performance on low power devices may be lower.
Overview

The Image Enhancement core performs two main functions: Noise Reduction and Edge Enhancement. Both of these functions use the edge information of the image not only for edge enhancement, but also to preserve edges during noise reduction.

First, an edge map is calculated which controls the strength and direction of noise reduction and/or edge enhancement that is applied to the image. After applying low-pass filters for noise reduction and high-pass filters for edge enhancement, the combined results are fed to two optional modules. The optional anti-halo module reduces the ringing or overshoot effect of high-pass filters. The optional anti-alias module reduces aliasing artifacts which can appear when edge enhancement is performed along with clipping/clamping.

Feature Summary

The Image Enhancement core offers noise reduction and/or edge enhancement. For edge enhancement, optional anti-halo and anti-alias post-processing modules are available to reduce image artifacts that can appear from the high-pass filtering of the edge enhancement filters. The amount of noise reduction and edge enhancement is controlled through user parameters. There are two variations of the algorithm offered to choose between high performance and minimal resource usage.

This core works on YCbCr 4:4:4 and 4:2:2 data. The core is capable of a maximum resolution of 7680 columns by 7680 rows with 8, 10, 12, or 16 bits per pixel and supports the bandwidth necessary for High-definition (1080p60) resolutions in all Xilinx FPGA device families. Higher resolutions can be supported in Xilinx high-performance device families.

You can configure and instantiate the core from Vivado® design tools. Core functionality may 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 and Ordering Information

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the Image Enhancement product web page at:

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.
Chapter 2

Product Specification

Standards

The Image Enhancement core is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. See the Video IP: AXI Feature Adoption section of the AXI Reference Guide [Ref 1] for additional information.

Performance

The following sections detail the performance characteristics of the Image Enhancement core.

Maximum Frequencies

This section contains 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 device, using a different version of Xilinx® tools and other factors. See Table 2-1 through Table 2-3 for device-specific information.

Latency

This section includes equations to calculate the latency of the core. A delay of one line is equal to the number of video clock cycles between subsequent EOL Signal pulses.

Noise Reduction Only

The latency through the noise reduction module is two lines + 22 clock cycles.

Edge Enhancement Only

The latency through the edge enhancement module is two lines + 20 clock cycles. Add three more clock cycles if the optional anti-aliasing is used.
Both Noise Reduction and Edge Enhancement

With the Minimal Resources architecture, the latency through the noise reduction and edge enhancement modules is two lines + 24 clock cycles. Add three more clock cycles if the optional anti-aliasing is used.

If the High Performance architecture is used, add two more lines + 11 more clock cycles of latency.

Throughput

The Image Enhancement 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 a number of clock cycles, during which the s_axis_video_tready is deasserted 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 deasserted signaling that the core is not ready to process samples.

When the core is processing timed streaming video (which is typical for most video systems), 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, for example, a frame buffer, the throughput of the core can be defined as follows (assuming a worst case latency of 38 clock cycles and 4 scan lines):

\[
R_{MAX} = f_{ACLK} \times \frac{ROWS}{ROWS + 4} \times \frac{COLS}{COLS + 38}
\]

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/s.

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

\[
f_{ACLK} = R_{MAX} \times \frac{ROWS + 4}{ROWS} \times \frac{COLS + 38}{COLS} = 124.4 \times \frac{1084}{1080} \times \frac{1958}{1920} = 127.3
\]

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.

The information presented in Table 2-1 through Table 2-3 is a guide to the resource utilization and maximum clock frequency of the Image Enhancement core for all input/output width combinations for Virtex®-7, Kintex®-7, Artix®-7, and Zynq®-7000 FPGA families using the Vivado® Design Suite. UltraScale™ results are expected to be similar to 7 series results. This core does not use any dedicated I/O or CLK resources. The design was tested with the 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: Kintex-7 FPGA and Zynq-7000 Devices with Kintex Based Programmable Logic Performance

<table>
<thead>
<tr>
<th>DATA WIDTH</th>
<th>HAS AXI4 LITE</th>
<th>VIDEO FORMAT</th>
<th>NOISE REDUCTION</th>
<th>EDGE ENHANCEMENT</th>
<th>HALO SUPPRESSION</th>
<th>ANTI-ALIASING</th>
<th>HIGH PERFORMANCE ALGORITHM</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAM36/18</th>
<th>DSP48s</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td></td>
<td>3471</td>
<td>2870</td>
<td>3803</td>
<td>11/0</td>
<td>21</td>
<td>266</td>
</tr>
<tr>
<td>10</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td></td>
<td>4128</td>
<td>3588</td>
<td>4437</td>
<td>12/1</td>
<td>21</td>
<td>266</td>
</tr>
<tr>
<td>12</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td></td>
<td>4727</td>
<td>4117</td>
<td>5088</td>
<td>15/1</td>
<td>23</td>
<td>172</td>
</tr>
<tr>
<td>16</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td></td>
<td>5763</td>
<td>5066</td>
<td>6356</td>
<td>20/1</td>
<td>23</td>
<td>172</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>3326</td>
<td>2797</td>
<td>3561</td>
<td>4/0</td>
<td>21</td>
<td>250</td>
</tr>
<tr>
<td>10</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>3902</td>
<td>3344</td>
<td>4153</td>
<td>4/1</td>
<td>21</td>
<td>242</td>
</tr>
<tr>
<td>12</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>4458</td>
<td>3857</td>
<td>4762</td>
<td>5/1</td>
<td>23</td>
<td>172</td>
</tr>
<tr>
<td>16</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>5328</td>
<td>4734</td>
<td>5946</td>
<td>7/1</td>
<td>23</td>
<td>172</td>
</tr>
<tr>
<td>8</td>
<td>no</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>2973</td>
<td>2574</td>
<td>3162</td>
<td>11/0</td>
<td>21</td>
<td>266</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:4:4</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td></td>
<td>4138</td>
<td>3499</td>
<td>4439</td>
<td>13/1</td>
<td>29</td>
<td>274</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>N/A</td>
<td>3087</td>
<td>2614</td>
<td>3344</td>
<td>4/0</td>
<td>16</td>
<td>226</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>N/A</td>
<td></td>
<td>2511</td>
<td>2037</td>
<td>2601</td>
<td>4/0</td>
<td>5</td>
<td>258</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>N/A</td>
<td></td>
<td>2660</td>
<td>2173</td>
<td>2809</td>
<td>4/0</td>
<td>7</td>
<td>250</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>N/A</td>
<td></td>
<td>2646</td>
<td>2206</td>
<td>2855</td>
<td>4/0</td>
<td>11</td>
<td>250</td>
</tr>
</tbody>
</table>
Table 2-2:  Artix-7 FPGA and Zynq-7000 Device with Artix Based Programmable Logic
Performance

<table>
<thead>
<tr>
<th>Data Width</th>
<th>Has AXI4 Lite</th>
<th>Video Format</th>
<th>Noise Reduction</th>
<th>Edge Enhancement</th>
<th>Halo Suppression</th>
<th>Anti Aliasing</th>
<th>High Performance Algorithm</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFS</th>
<th>RAM36/18</th>
<th>DSP48s</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>3476</td>
<td>2874</td>
<td>3803</td>
<td>11/0</td>
<td>21</td>
<td>180</td>
</tr>
<tr>
<td>10</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>4242</td>
<td>3590</td>
<td>4437</td>
<td>12/1</td>
<td>21</td>
<td>188</td>
</tr>
<tr>
<td>12</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>4781</td>
<td>4116</td>
<td>5088</td>
<td>15/1</td>
<td>23</td>
<td>140</td>
</tr>
<tr>
<td>16</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>5823</td>
<td>5065</td>
<td>6356</td>
<td>20/1</td>
<td>23</td>
<td>140</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>3345</td>
<td>2786</td>
<td>3561</td>
<td>4/0</td>
<td>21</td>
<td>180</td>
</tr>
<tr>
<td>10</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>3800</td>
<td>3329</td>
<td>4153</td>
<td>4/1</td>
<td>21</td>
<td>172</td>
</tr>
<tr>
<td>12</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>4483</td>
<td>3857</td>
<td>4762</td>
<td>5/1</td>
<td>23</td>
<td>148</td>
</tr>
<tr>
<td>16</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>5378</td>
<td>4735</td>
<td>5946</td>
<td>7/1</td>
<td>23</td>
<td>140</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>3051</td>
<td>2565</td>
<td>3162</td>
<td>11/0</td>
<td>21</td>
<td>164</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:4:4</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>4177</td>
<td>3504</td>
<td>4439</td>
<td>13/1</td>
<td>29</td>
<td>188</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>N/A</td>
<td>3108</td>
<td>2596</td>
<td>3344</td>
<td>4/0</td>
<td>16</td>
<td>172</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>N/A</td>
<td>2497</td>
<td>2027</td>
<td>2601</td>
<td>4/0</td>
<td>5</td>
<td>180</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>N/A</td>
<td>2667</td>
<td>2164</td>
<td>2809</td>
<td>4/0</td>
<td>7</td>
<td>188</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>N/A</td>
<td>2667</td>
<td>2194</td>
<td>2855</td>
<td>4/0</td>
<td>11</td>
<td>180</td>
</tr>
</tbody>
</table>

Table 2-3:  Virtex-7 FPGA Performance

<table>
<thead>
<tr>
<th>Data Width</th>
<th>Has AXI4 Lite</th>
<th>Video Format</th>
<th>Noise Reduction</th>
<th>Edge Enhancement</th>
<th>Halo Suppression</th>
<th>Anti Aliasing</th>
<th>High Performance Algorithm</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFS</th>
<th>RAM36/18</th>
<th>DSP48s</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>3479</td>
<td>2870</td>
<td>3803</td>
<td>11/0</td>
<td>21</td>
<td>266</td>
</tr>
<tr>
<td>10</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>4192</td>
<td>3587</td>
<td>4437</td>
<td>12/1</td>
<td>21</td>
<td>266</td>
</tr>
<tr>
<td>12</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>4712</td>
<td>4117</td>
<td>5088</td>
<td>15/1</td>
<td>23</td>
<td>172</td>
</tr>
<tr>
<td>16</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>5761</td>
<td>5066</td>
<td>6356</td>
<td>20/1</td>
<td>23</td>
<td>172</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>3303</td>
<td>2800</td>
<td>3561</td>
<td>4/0</td>
<td>21</td>
<td>258</td>
</tr>
<tr>
<td>10</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>3936</td>
<td>3346</td>
<td>4153</td>
<td>4/1</td>
<td>21</td>
<td>250</td>
</tr>
</tbody>
</table>
Core Interfaces and Register Space

Port Descriptions

The Image Enhancement 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 Enhancement 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 using 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.

Table 2-3: Virtex-7 FPGA Performance (Cont’d)

<table>
<thead>
<tr>
<th>DATA WIDTH</th>
<th>HAS AXI4 LITE</th>
<th>VIDEO FORMAT</th>
<th>NOISE REDUCTION</th>
<th>EDGE ENHANCEMENT</th>
<th>HALO SUPPRESSION</th>
<th>ANTI ALIASING</th>
<th>HIGH PERFORMANCE ALGORITHM</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAM36/18</th>
<th>DSP48s</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>4503</td>
<td>3857</td>
<td>4762</td>
<td>5/1</td>
<td>23</td>
<td>172</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>5558</td>
<td>4736</td>
<td>5946</td>
<td>7/1</td>
<td>23</td>
<td>172</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>no</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>2974</td>
<td>2572</td>
<td>3162</td>
<td>11/0</td>
<td>21</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:4:4</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>4153</td>
<td>3500</td>
<td>4439</td>
<td>13/1</td>
<td>29</td>
<td>266</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>N/A</td>
<td>3113</td>
<td>2613</td>
<td>3344</td>
<td>4/0</td>
<td>16</td>
<td>266</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>N/A</td>
<td>2525</td>
<td>2039</td>
<td>2601</td>
<td>4/0</td>
<td>5</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>N/A</td>
<td>2651</td>
<td>2174</td>
<td>2809</td>
<td>4/0</td>
<td>7</td>
<td>266</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>4:2:2</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>N/A</td>
<td>2675</td>
<td>2204</td>
<td>2855</td>
<td>4/0</td>
<td>11</td>
<td>266</td>
<td></td>
</tr>
</tbody>
</table>
Common Interface Signals

Table 2-4 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-4: Common Interface Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACLK</td>
<td>In</td>
<td>1</td>
<td>Video Core Clock</td>
</tr>
<tr>
<td>ACLKEN</td>
<td>In</td>
<td>1</td>
<td>Video Core Active High Clock Enable</td>
</tr>
<tr>
<td>ARESETn</td>
<td>In</td>
<td>1</td>
<td>Video Core Active Low Synchronous Reset</td>
</tr>
<tr>
<td>INTC_IF</td>
<td>Out</td>
<td>9</td>
<td>Optional External Interrupt Controller Interface. Available only when INTC_IF is selected on GUI.</td>
</tr>
<tr>
<td>IRQ</td>
<td>Out</td>
<td>1</td>
<td>Optional Interrupt Request Pin. Available only when AXI4-Lite interface is selected on GUI.</td>
</tr>
</tbody>
</table>
The **ACLK**, **ACLKEN** and **ARESETn** signals are shared between the core and the AXI4-Stream data interfaces. The AXI4-Lite control interface has its own set of clock, clock enable and reset pins: **S_AXI_ACLK**, **S_AXI_ACLKEN** and **S_AXI_ARESETn**. See The Interrupt Subsystem for a detailed description of the **INTC_IF** and **IRQ** pins.

### ACLK

The AXI4-Stream interface must be synchronous to the core clock signal **ACLK**. All AXI4-Stream interface input signals are sampled on the rising edge of **ACLK**. All AXI4-Stream output signal changes occur after the rising edge of **ACLK**. The AXI4-Lite interface is unaffected by the **ACLK** signal.

### ACLKEN

The **ACLKEN** pin is an active-high, synchronous clock-enable input pertaining to AXI4-Stream interfaces. Setting **ACLKEN** low (deasserted) 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 deasserted, core inputs are not sampled, except **ARESETn**, which supersedes **ACLKEN**. The AXI4-Lite interface is unaffected by the **ACLKEN** signal.

### ARESETn

The **ARESETn** pin is an active-Low, synchronous reset input pertaining to only AXI4-Stream interfaces. **ARESETn** supersedes **ACLKEN**, and when set to 0, the core resets at the next rising edge of **ACLK** even if **ACLKEN** is deasserted. The **ARESETn** signal must be synchronous to the **ACLK** and must be held low for a minimum of 32 clock cycles of the slowest clock. The AXI4-Lite interface is unaffected by the **ARESETn** signal.

### Data Interface

The Image Enhancement core receives and transmits data using AXI4-Stream interfaces that implement a video protocol as defined in the Video IP: AXI Feature Adoption section of the Vivado AXI Reference Guide (UG1037) [Ref 1].

### AXI4-Stream Signal Names and Descriptions

**Table 2-5** describes the AXI4-Stream signal names and descriptions.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_axis_video_tdata</td>
<td>In</td>
<td>16, 24, 32, 40, 48</td>
<td>Input Video Data</td>
</tr>
<tr>
<td>s_axis_video_tvalid</td>
<td>In</td>
<td>1</td>
<td>Input Video Valid Signal</td>
</tr>
<tr>
<td>s_axis_video_tready</td>
<td>Out</td>
<td>1</td>
<td>Input Ready</td>
</tr>
</tbody>
</table>
The AXI4-Stream interface specification restricts TDATA widths to integer multiples of 8 bits. Therefore, 10 and 12 bit data must be padded with zeros on the MSB to form a 24-, 32-, or 40-bit wide vectors before connecting to s_axis_video_tdata. Padding does not affect the size of the core.

Similarly, YCbCr data on the output m_axis_video_tdata is packed and padded to multiples of 8 bits as necessary, as seen in Figure 2-2 and Figure 2-3. Zero padding the most significant bits is only necessary for 10- and 12-bit wide data.

Table 2-5: AXI4-Stream Data Interface Signal Descriptions (Cont’d)

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_axis_video_tuser</td>
<td>In</td>
<td>1</td>
<td>Input Video Start Of Frame</td>
</tr>
<tr>
<td>s_axis_video_tlast</td>
<td>In</td>
<td>1</td>
<td>Input Video End Of Line</td>
</tr>
<tr>
<td>m_axis_video_tdata</td>
<td>Out</td>
<td>16, 24, 32, 40, 48</td>
<td>Output Video Data</td>
</tr>
<tr>
<td>m_axis_video_tvalid</td>
<td>Out</td>
<td>1</td>
<td>Output Valid</td>
</tr>
<tr>
<td>m_axis_video_tready</td>
<td>In</td>
<td>1</td>
<td>Output Ready</td>
</tr>
<tr>
<td>m_axis_video_tuser</td>
<td>Out</td>
<td>1</td>
<td>Output Video Start Of Frame</td>
</tr>
<tr>
<td>m_axis_video_tlast</td>
<td>Out</td>
<td>1</td>
<td>Output Video End Of Line</td>
</tr>
</tbody>
</table>

Video Data

The AXI4-Stream interface specification restricts TDATA widths to integer multiples of 8 bits. Therefore, 10 and 12 bit data must be padded with zeros on the MSB to form a 24-, 32-, or 40-bit wide vectors before connecting to s_axis_video_tdata. Padding does not affect the size of the core.

Similarly, YCbCr data on the output m_axis_video_tdata is packed and padded to multiples of 8 bits as necessary, as seen in Figure 2-2 and Figure 2-3. Zero padding the most significant bits is only necessary for 10- and 12-bit wide data.

Figure 2-2: 10-Bit YCbCr Data Encoding for 4:4:4 on m_axis_video_tdata

Figure 2-3: 10-Bit YCbCr Data Encoding for 4:2:2 on s_axis_video_tdata and m_axis_video_tdata

YCbCr data is packed on the s_axis_video_tdata and m_axis_video_tdata busses as shown in Figure 2-4 and Figure 2-5. For 4:4:4 chroma format, Y, Cb, and Cr are on a single bus and run at full sample rate, as shown in Figure 2-4.
Figure 2-4: YCbCr 4:4:4

For 4:2:2, Cb and Cr are interleaved on the s_axis_video_tdata and m_axis_video_tdata busses. The first active video data sample contains Cb first, as shown in Figure 2-5.

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 2-6. 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 Enhancement core driving s_axis_video_tready) may change value until the transaction completes (s_axis_video_tready and s_axis_video_tvalid and ACLKEN are high on the rising edge of ACLK). Once asserted, s_axis_video_tvalid may only be deasserted 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 deasserted or remain asserted to initiate a new transfer.
Guidelines on Driving \texttt{m_axis_video_tready}

The \texttt{m_axis_video_tready} signal can be asserted before, during or after the cycle in which the Image Enhancement core asserted \texttt{m_axis_video_tvalid}. The assertion of \texttt{m_axis_video_tready} may be dependent on the value of \texttt{m_axis_video_tvalid}. A slave that can immediately accept data qualified by \texttt{m_axis_video_tvalid}, should pre-assert its \texttt{m_axis_video_tready} signal until data is received. Alternatively, \texttt{m_axis_video_tready} can be registered and driven the cycle following VALID assertion.

\textbf{RECOMMENDED:} The AXI4-Stream slave should drive READY independently, or pre-assert READY to minimize latency.

Start of Frame Signals - \texttt{m_axis_video_tuser}, \texttt{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 2-6. 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 - \texttt{m_axis_video_tlast}, \texttt{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 2-7.
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 using the 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 interface the core to a particular image sensor with a known, stationary resolution and use constant enhancement parameters. In constant configuration the image resolution (number of active pixels per scan line and the number of active scan lines per frame) and the enhancement parameters are hard coded into the core through the Image Enhancement 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 Enhancement core can be controlled through the AXI4-Lite interface using read and write transactions to the Image Enhancement register space.

Table 2-6: AXI4-Lite Interface Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_axi_aclk</td>
<td>In</td>
<td>1</td>
<td>AXI4-Lite clock</td>
</tr>
<tr>
<td>s_axi_aclken</td>
<td>In</td>
<td>1</td>
<td>AXI4-Lite clock enable</td>
</tr>
<tr>
<td>s_axi_aresetn</td>
<td>In</td>
<td>1</td>
<td>AXI4-Lite synchronous Active-Low reset</td>
</tr>
</tbody>
</table>
Table 2-6: AXI4-Lite Interface Signals (Cont’d)

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_axi_awvalid</td>
<td>In</td>
<td>1</td>
<td>AXI4-Lite Write Address Channel Write Address Valid.</td>
</tr>
<tr>
<td>s_axi_awread</td>
<td>Out</td>
<td>1</td>
<td>AXI4-Lite Write Address Channel Write Address Ready. Indicates DMA ready to accept the write address.</td>
</tr>
<tr>
<td>s_axi_awaddr</td>
<td>In</td>
<td>32</td>
<td>AXI4-Lite Write Address Bus</td>
</tr>
<tr>
<td>s_axi_wvalid</td>
<td>In</td>
<td>1</td>
<td>AXI4-Lite Write Data Channel Write Data Valid.</td>
</tr>
<tr>
<td>s_axi_wready</td>
<td>Out</td>
<td>1</td>
<td>AXI4-Lite Write Data Channel Write Data Ready. Indicates DMA is ready to accept the write data.</td>
</tr>
<tr>
<td>s_axi_wdata</td>
<td>In</td>
<td>32</td>
<td>AXI4-Lite Write Data Bus</td>
</tr>
<tr>
<td>s_axi_bresp</td>
<td>Out</td>
<td>2</td>
<td>AXI4-Lite Write Response Channel. Indicates results of the write transfer.</td>
</tr>
<tr>
<td>s_axi_bvalid</td>
<td>Out</td>
<td>1</td>
<td>AXI4-Lite Write Response Channel Response Valid.</td>
</tr>
<tr>
<td>s_axi_bready</td>
<td>In</td>
<td>1</td>
<td>AXI4-Lite Write Response Channel Ready. Indicates target is ready to receive response.</td>
</tr>
<tr>
<td>s_axi_arvalid</td>
<td>In</td>
<td>1</td>
<td>AXI4-Lite Read Address Channel Read Address Valid</td>
</tr>
<tr>
<td>s_axi_arready</td>
<td>Out</td>
<td>1</td>
<td>Ready. Indicates DMA is ready to accept the read address.</td>
</tr>
<tr>
<td>s_axi_araddr</td>
<td>In</td>
<td>32</td>
<td>AXI4-Lite Read Address Bus</td>
</tr>
<tr>
<td>s_axi_rvalid</td>
<td>Out</td>
<td>1</td>
<td>AXI4-Lite Read Data Channel Read Data Valid</td>
</tr>
<tr>
<td>s_axi_rready</td>
<td>In</td>
<td>1</td>
<td>AXI4-Lite Read Data Channel Read Data Ready. Indicates target is ready to accept the read data.</td>
</tr>
<tr>
<td>s_axi_rdata</td>
<td>Out</td>
<td>32</td>
<td>AXI4-Lite Read Data Bus</td>
</tr>
<tr>
<td>s_axi_rresp</td>
<td>Out</td>
<td>2</td>
<td>AXI4-Lite Read Response Channel Response. Indicates results of the read transfer.</td>
</tr>
</tbody>
</table>

**S_AXI_ACLK**

The AXI4-Lite interface must be synchronous to the **S_AXI_ACLK** clock signal. The AXI4-Lite interface input signals are sampled on the rising edge of **ACLK**. The AXI4-Lite output signal changes occur after the rising edge of **ACLK**. The AXI4-Stream interfaces signals are not affected by the **S_AXI_ACLK**.

**S_AXI_ACLKEN**

The **S_AXI_ACLKEN** pin is an active-High, synchronous clock-enable input for the AXI4-Lite interface. Setting **S_AXI_ACLKEN** low (deasserted) halts the operation of the AXI4-Lite interface despite rising edges on the **S_AXI_ACLK** pin. AXI4-Lite interface states are maintained, and AXI4-Lite interface output signal levels are held until **S_AXI_ACLKEN** is asserted again. When **S_AXI_ACLKEN** is deasserted, AXI4-Lite interface inputs are not sampled, except **S_AXI_ARESETn**, which supersedes **S_AXI_ACLKEN**. The AXI4-Stream interfaces signals are not affected by the **S_AXI_ACLKEN**.
**S_AXI_ARESETn**

The **S_AXI_ARESETn** pin is an active-Low, synchronous reset input for the AXI4-Lite interface. **S_AXI_ARESETn** supersedes **S_AXI_ACLKEN**, and when set to 0, the core resets at the next rising edge of **S_AXI_ACLK** even if **S_AXI_ACLKEN** is deasserted. The **S_AXI_ARESETn** signal must be synchronous to the **S_AXI_ACLK** and must be held low for a minimum of 32 clock cycles of the slowest clock. The **S_AXI_ARESETn** input is resynchronized to the **ACLK** clock domain. The AXI4-Stream interfaces and core signals are also reset by **S_AXI_ARESETn**.

**Register Space**

The standardized Xilinx Video IP register space is partitioned into control-, timing-, and core specific registers. The Image Enhancement core uses only one timing related register, **ACTIVE_SIZE** (0x0020), which allows specifying the input frame dimensions. Also, the core has three core-specific register, **NOISE_THRESHOLD** (0x0100), **ENHANCE_STRENGTH** (0x0104), and **HALO_SUPPRESS** (0x0108) which allows specifying the strength of the enhancement effects.

<table>
<thead>
<tr>
<th>Address (hex) BASEADDR +</th>
<th>Register Name</th>
<th>Access Type</th>
<th>Double Buffered</th>
<th>Default Value</th>
<th>Register Description</th>
</tr>
</thead>
</table>
| 0x0000 CONTROL            | R/W           | No          | 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         | 0x08000000      | 7-0: REVISION_NUMBER
11-8: PATCH_ID
15-12: VERSION_REVISION
23-16: VERSION_MINOR
31-24: VERSION_MAJOR |
### Table 2-7: Register Names and Descriptions (Cont’d)

<table>
<thead>
<tr>
<th>Address (hex) BASEADDR</th>
<th>Register Name</th>
<th>Access Type</th>
<th>Double Buffered</th>
<th>Default Value</th>
<th>Register Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0014</td>
<td>SYSDEBUG0</td>
<td>R</td>
<td>N/A</td>
<td>0</td>
<td>0-31: Frame Throughput monitor(1)</td>
</tr>
<tr>
<td>0x0018</td>
<td>SYSDEBUG1</td>
<td>R</td>
<td>N/A</td>
<td>0</td>
<td>0-31: Line Throughput monitor(1)</td>
</tr>
<tr>
<td>0x001C</td>
<td>SYSDEBUG2</td>
<td>R</td>
<td>N/A</td>
<td>0</td>
<td>0-31: Pixel Throughput monitor(1)</td>
</tr>
<tr>
<td>0x0020</td>
<td>ACTIVE_SIZE</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified through GUI in Frame Dimensions</td>
<td></td>
</tr>
<tr>
<td>0x0100</td>
<td>NOISE_THRESHOLD</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified through GUI in the Noise Threshold text box</td>
<td>Allowed values are 0 to 2^{DATA_WIDTH}-1</td>
</tr>
<tr>
<td>0x0104</td>
<td>ENHANCE_STRENGTH</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified through GUI in the Enhance Strength text box</td>
<td>Allowed values are 0 to 1 represented by 16 bits with 15 fractional bits</td>
</tr>
<tr>
<td>0x0108</td>
<td>HALO_SUPPRESS</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified through GUI in the Halo Suppress text box</td>
<td>Allowed values are 0 to 1 represented by 16 bits with 15 fractional bits</td>
</tr>
</tbody>
</table>

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 Enhancement core `ACTIVE_SIZE`, `NOISE_THRESHOLD`, `ENHANCE_STRENGTH`, and `HALO_SUPPRESS` 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 are copied over to the active set at the end of the AXI4-Stream frame, if and only if \texttt{REG\_UPDATE} is set. Setting \texttt{REG\_UPDATE} to 0 before updating multiple register values, then setting \texttt{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 \texttt{CONTROL} register, \texttt{BYPASS}, switches the core to bypass mode if debug features are enabled. In bypass mode the Image Enhancement core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output. See Debug Tools 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 \texttt{CONTROL} register, \texttt{TEST\_PATTERN}, switches the core to test-pattern generator mode if debug features are enabled. See Debug Tools 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 \texttt{CONTROL} register, \texttt{FRAME\_SYNC\_RESET} and \texttt{SW\_RESET} facilitate software reset. Setting \texttt{SW\_RESET} reinitializes the core to GUI default values, all internal registers and outputs are cleared and held at initial values until \texttt{SW\_RESET} is set to 0. The \texttt{SW\_RESET} flag is not synchronized with the AXI4-Stream interfaces. Resetting the core while frame processing is in progress causes image tearing. For applications where the software reset functionality is desirable, but image tearing has to be avoided a frame synchronized software reset (\texttt{FRAME\_SYNC\_RESET}) is available. Setting \texttt{FRAME\_SYNC\_RESET} to 1 resets the core at the end of the frame being processed, or immediately if the core is between frames when the \texttt{FRAME\_SYNC\_RESET} was asserted. After reset, the \texttt{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 \texttt{RESET} bits is 0. Core instances with no AXI4-Lite control interface can only be reset using the \texttt{ARESETn} pin.

\textbf{STATUS (0x0004) Register}

All bits of the \texttt{STATUS} register can be used to request an interrupt from the host processor.

\begin{itemize}
  \item \textbf{IMPORTANT:} Bits of the \texttt{STATUS} register remain set until cleared.
\end{itemize}

Bits of the \texttt{STATUS} register remain set after an event associated with the particular \texttt{STATUS} register bit; even if the event condition is not present at the time the interrupt is serviced. This is to facilitate identification of the interrupt source.

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

Bit 0 of the \texttt{STATUS} register, \texttt{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 (0x00008) 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 (0x0000C) Register**

Any bits of the `STATUS` register can generate a host-processor interrupt request via the IRQ pin. The Interrupt Enable register helps select the bits of `STATUS` register that 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 (0x00010) 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. See Table 2-7 for details.

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. See Appendix C, Debugging 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. See Appendix C, Debugging 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. See Appendix C, Debugging 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.

NOISE_THRESHOLD (0x0100) Register

The NOISE_THRESHOLD register contains the amount of noise reduction applied by the low-pass filters. Allowed values are from 0 to 2^DATA_WIDTH-1. See Chapter 3, Designing with the Core for more information on defining the NOISE_THRESHOLD value.

ENHANCE_STRENGTH (0x0104) Register

The ENHANCE_STRENGTH register contains the amount of edge enhancement applied by the high-pass filters. The allowed values are from 0 to 32768 which is the integer representation of the range 0 to 1 using 16 bits with 15 fractional bits. Multiplication by
2^15 yields the integer representation. See Chapter 3, Designing with the Core for more information on defining the ENHANCE_STRENGTH value.

**HALO_SUPPRESS (0x0108) Register**

The HALO_SUPPRESS register contains the amount of halo suppression. The allowed values are from 0 to 32768 which is the integer representation of the range 0 to 1 using 16 bits with 15 fractional bits. Multiplication by 2^15 yields the integer representation. See Chapter 3, Designing with the Core for more information on defining the HALO_SUPPRESS value.

**The 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 Enhancement 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 2-8.

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

**Table 2-8: INTC_IF Signal Functions**

<table>
<thead>
<tr>
<th>INTC_IF signal</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Frame processing start</td>
</tr>
<tr>
<td>1</td>
<td>Frame processing complete</td>
</tr>
<tr>
<td>2</td>
<td>Pixel counter terminal count</td>
</tr>
<tr>
<td>3</td>
<td>Line counter terminal count</td>
</tr>
<tr>
<td>4</td>
<td>Slave Error</td>
</tr>
<tr>
<td>5</td>
<td>EOL Early</td>
</tr>
<tr>
<td>6</td>
<td>EOL Late</td>
</tr>
</tbody>
</table>
In a system integration tool, 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.

**Table 2-8: INTC_IF Signal Functions (Cont’d)**

<table>
<thead>
<tr>
<th>INTC_IF signal</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SOF Early</td>
</tr>
<tr>
<td>8</td>
<td>SOF Late</td>
</tr>
</tbody>
</table>
Designing with the Core

The Image Enhancement core offers noise reduction and edge enhancement functionality in one core. The core can perform either noise reduction or edge enhancement or both.

Two algorithms are available to choose between High Performance and Minimal Resource Usage.

The High Performance algorithm is a pipelined solution which produces results with higher visual quality. First, noise reduction is performed on the image, and then the noise reduced image is passed to the edge enhancement portion.

The Minimal Resources algorithm is a parallel solution and provides a smaller footprint as it uses fewer line buffers. Noise reduction and edge enhancement are both performed on the input image, and the results are blended together.

Figure 3-1: High Performance Algorithm

The Minimal Resources algorithm is a parallel solution and provides a smaller footprint as it uses fewer line buffers. Noise reduction and edge enhancement are both performed on the input image, and the results are blended together.
In both algorithms, first an edge map is calculated, which controls the strength and direction of noise reduction and/or edge enhancement that is applied to the image. After applying low-pass filters for noise reduction and high-pass filters for edge enhancement, the combined results are fed to two optional modules. The optional anti-halo module reduces the ringing or overshoot effect of high-pass filters. The optional anti-alias module reduces aliasing artifacts which can appear when edge-enhancement is performed along with clipping/clamping.

**Edge Map Calculation**

The first step is to build a coherent edge map of the image. Edge information is used to set the strength and direction of the two dimensional noise reduction and edge enhancement filters.

Edge map calculation takes place in two steps.

1. Two dimensional FIR filters are used to extract edge content in four directions: horizontally, vertically and along both diagonals.

2. Using elongated, perpendicular structural elements and morphological processing, edge content is analyzed to provide a smooth, non-ambiguous map of local edge directionality. The purpose of this step is to provide a cleaner definition of the edges in each direction.
Noise Reduction

The noise reduction module constructs a pixel specific blurring kernel to perform adaptive filtering on specific neighborhoods. The kernel is constructed using Gaussian-like directional low-pass filters. The Edge Map and Noise Threshold parameters are used to compute the gains for blending the outputs of the four directional blurring filters with the original image. By using the Edge Map information, the algorithm can avoid blurring across edges. Near edges, blurring is only applied in the same direction as the edge.

\[
1 - 0.25^*[(1-kH) + (1-kV) + (1-kL) + (1-kR)]
\]

Figure 3-3: Noise Reduction Module

Edge Enhancement

The edge enhancement algorithm uses direction specific Laplacian filters to enhance image edges. The filter is applied based on direction specific edge information calculated in the Edge Map. Combining the directed Laplacian operators using the edge content information as weights creates an edge enhancement operator which is oriented to be directly
perpendicular to the edge, resulting in optimal use of the kernel size and minimizing noise amplification.

**Figure 3-4: Edge Enhancement Module**

### Optional Post-Processing Modules

#### Anti-Halo

To prevent overshoot and undershoot at enhanced edges, an additional processing step can be added to the algorithm. During this step each enhanced pixel value is compared to the corresponding pixel and its neighborhood in the original image. The edge enhanced pixel value is clipped at the maximum pixel value of its 8 nearest neighbors from the original frame. Similarly, each enhanced pixel value is clamped at the minimum pixel value of its 8 nearest neighbors of the original image.
Anti-Aliasing

When enhancing already sharp edges running at a low angle in reference to the sampling grid, edge enhancement may introduce aliasing artifacts due to saturation at the extremes of the dynamic range permitted by the pixel data representation (Figure 3-6).

Similar to adaptive edge enhancement, edge adaptive low-pass filters are used to reduce the aliasing artifacts introduced. The anti-aliasing filters are only applied along the edges so as not to blur the edge. The anti-aliasing filters use the same edge map information used for the Laplacian operators for edge enhancement.

Defining Gains

Noise Threshold

The amount of noise reduction can be controlled through the programmable Noise Threshold parameter. The threshold has a range of 0 to $2^{\text{DATA WIDTH}}-1$. 
Areas of the image above the threshold have the blurring filter applied. This means a Noise Threshold=0 does not apply any noise reduction filtering, and the output of the noise module is identical to the input. A Noise Threshold=2^DATA_WIDTH-1 applies the blurring filters across the entire image (weighted by the edge map to preserve edges).

**Enhance Strength**

The amount of edge enhancement can be controlled through the programmable Enhance Strength parameter. The strength parameter has a range of 0 to 1. This is represented as an unsigned 16 bit number with 15 fractional bits. To get the integer equivalent, multiply by 2^15. For example, a value of 1 is represented as 32768.

The larger the strength, the stronger the edge enhancement. An Enhance Strength=0 does not apply any edge enhancement, and the output of the edge enhancement module is identical to the input. Setting the Enhance Strength to a large value can produce visual artifacts. These effects can be reduced with the optional anti-halo and anti-aliasing modules.

**Halo Suppress**

The high pass filter of the edge enhancement can add a halo artifact to the image. The anti-halo module can remove the halo effect. The amount of halo suppression can be controlled through the programmable Halo Suppress parameter. The halo suppression parameter has a range of 0 to 1. This is represented as an unsigned 16 bit number with 15 fractional bits. To get the integer equivalent, multiply by 2^15. For example, a value of 1 is represented as 32768.

The larger the halo suppression value, the stronger the halo suppression. A Halo Suppress=0 does not perform any halo suppression, and the output of the anti-halo module is identical to the input. Setting Halo Suppress=1 completely suppresses any halo by limiting each pixel value to the minimum and maximum value of its neighborhood in the original image.

**Guidelines for Defining Gains**

One method of determining what gain values to use is to start with no image enhancement and then increase the control values one at a time until the desired effect is achieved. A user can start by setting Noise Threshold, Enhance Strength and Halo Suppress to zero. First, if noise removal is chosen, the Noise Threshold value should be increased starting from 0 until the noise removal is sufficient. This may blur the edges, but the edge enhancement can correct that. The Enhance Strength would be the next value to determine (assuming the edge enhancement feature is selected). This value can be slowly increased from 0 to 1 until the desired edge enhancement level is achieved. If a halo or aliasing effect appear, it can be reduced with the anti-halo and anti-aliasing processing. Once the Enhance Strength is set, the user can choose to add halo suppression. The Halo Suppress value can be increased from 0 to 1 until the desired halo reduction is reached. At this point, the anti-aliasing filters
can be added if preferred. Fine-tuning of the gain values can be done at this point until the desired visual effect is attained.

**General Design Guidelines**

The Image Enhancement 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 Enhancement block cannot change the input/output image sizes, the input and output pixel clock rates, or the frame rate.

**RECOMMENDED:** This core should be 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 image sensor. The Video In to AXI4-Stream IP core converts the incoming video data stream to AXI4-Stream Video Protocol.

Typically, the Image Enhancement core is part of an Image Sensor Pipeline (ISP) System, as shown in **Figure 3-7**.

![Figure 3-7: Image Sensor Pipeline System with Image Enhancement Core](image.png)
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 3-8.

S_AXI_ACLK

The AXI4-Lite interface uses the S_AXI_ACLK pin as its clock source. The ACLK pin is not shared between the AXI4-Lite and AXI4-Stream interfaces. The Image Enhancement core contains clock-domain crossing logic between the ACLK (AXI4-Stream and Video Processing) and S_AXI_ACLK (AXI4-Lite) clock domains. The core automatically ensures that the AXI4-Lite transactions completes even if the video processing is stalled with ARESETn, ACLKEN or with the video clock not running.

ACLKEN

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

ACLKEN may not be 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
Clock, Enable, and Reset Considerations

**IMPORTANT:** 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 2-2).

**IMPORTANT:** When two cores connected through 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 through the AXI4-Stream Interconnect or AXI-FIFO cores to avoid data corruption (Figure 2-3).

**S_AXI_ACLKEN**

The S_AXI_ACLKEN is the clock enable signal for the AXI4-Lite interface only. Driving this signal Low only affects the AXI4-Lite interface and does not halt the video processing in the ACLK clock domain.

**ARESETn**

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

**IMPORTANT:** ARESETn is not synchronized internally to AXI4-Stream frame processing. Deasserting ARESETn while a frame is being process leads to image tearing.

The external reset pulse needs to be held for 32 ACLK cycles to reset the core. The ARESETn signal only resets the AXI4-Stream interfaces. The AXI4-Lite interface is unaffected by the ARESETn signal to allow the video processing core to be reset without halting the AXI4-Lite interface.

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

**S_AXI_ARESETn**

The S_AXI_ARESETn signal is synchronous to the S_AXI_ACLK clock domain, but is internally synchronized to the ACLK clock domain. The S_AXI_ARESETn signal resets the entire core including the AXI4-Lite and AXI4-Stream interfaces.
System Considerations

The Image Enhancement core must 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 gathers 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 Enhancement, with the appropriate image dimensions.

If the target system uses only one configuration of the Image Enhancement core (i.e. does not need to be reprogrammed), you may choose to create a constant configuration by removing the AXI4-Lite interface. This reduces the core Slice footprint.

Clock Domain Interaction

The \texttt{ARESETn} and \texttt{ACLKEN} input signals do not reset or halt the AXI4-Lite interface. This allows the video processing to be reset or halted separately from the AXI4-Lite interface without disrupting AXI4-Lite transactions.

The AXI4-Lite interface responds with an error if the core registers cannot be read or written within 128 \texttt{S\_AXI\_ACLK} clock cycles. The core registers cannot be read or written if the \texttt{ARESETn} signal is held low, if the \texttt{ACLKEN} signal is held low or if the \texttt{ACLK} signal is not connected or not running. If core register read does not complete, the AXI4-Lite read transaction responds with \texttt{10} on the \texttt{S\_AXI\_RRESP} bus. Similarly, if a core register write does not complete, the AXI4-Lite write transaction responds with \texttt{10} on the \texttt{S\_AXI\_BRESP} bus. The \texttt{S\_AXI\_ARESETn} input signal resets the entire core.

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 Video IP cores are disabled/reset from system output towards the system input, and programmed/enabled from system input to system output. \texttt{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 and based on the error propagation and recovery settings, the Image Enhancement IP core can predict the expected beginning of the next frame. 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.
Chapter 4

Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core in the Vivado® Design Suite environment.

Vivado Integrated Design Environment (IDE)

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click on the selected IP or select the Customize IP command from the toolbar or popup menu.

For details, see the sections, “Working with IP” and “Customizing IP for the Design” in the Vivado Design Suite User Guide: Designing with IP (UG896) [Ref 3] and the “Working with the Vivado IDE” section in the Vivado Design Suite User Guide: Getting Started (UG910) [Ref 5].

If you are customizing and generating the core in the Vivado IP Integrator, see the Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator (UG994) [Ref 7] for detailed information. IP Integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the validate_bd_design command in the tcl console.

Note: Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version.

Interface

The Image Enhancement core is easily configured to the user’s specific needs through the Vivado IDE. This section provides a quick reference to the parameters that can be configured at generation time. Figure 4-1 shows the main Image Enhancement 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_enhance_v8_0` cannot be used as a component name.

- **Video Component Width**: Specifies the bit width of input samples. Permitted values are 8, 10, 12, and 16 bits. When using Vivado IP Integrator this parameter is automatically computed based on the Video Component Width of the Video IP core connected to the slave AXI-Stream video interface.

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

**IMPORTANT:** Debugging features are only available when the AXI4-Lite Register Interface is selected.

- INTC Interface: When selected, the core generates the optional \texttt{INTC\_IF} port, which gives parallel access to signals indicating frame processing status and error conditions. For more information, see The Interrupt Subsystem in Chapter 2.

- Input Format: Select the chroma format. The supported formats are YCbCr 4:4:4 and 4:2:2. When using Vivado IP Integrator this parameter is automatically computed based on the Video Format of the Video IP core connected to the slave AXI-Stream video interface.

- Image Enhancement Options: At least one of Image Noise Reduction or Image Edge Enhancement must be selected.
  - Image Noise Reduction: Selecting this checkbox adds image noise reduction functionality to the core.
    - Noise Threshold: This parameter controls the amount of noise reduction. Valid values are integers from 0 to $2^{\text{Video\_Component\_Width}}-1$.
  - Image Edge Enhancement: Selecting this checkbox adds image edge enhancement functionality to the core.
    - Enhance Strength: This parameter controls the amount of edge enhancement. Valid values are floating point numbers from 0 to 1.
  - Halo Suppression: With Image Edge Enhancement, selecting this checkbox adds halo suppression functionality to the core.
    - Halo Suppress: This parameter controls the amount of halo suppression. Valid values are floating point numbers from 0 to 1.
  - Anti-Alias Filtering: With Image Edge Enhancement, selecting this checkbox adds anti-aliasing functionality to the core.

- Architecture: When performing both noise reduction and edge enhancement, two architectures are available: High Performance offers better visual image quality, and Minimal Resource offers a smaller implementation footprint.

- Input Frame Dimensions:
  - Number of Pixels per Scanline: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the GUI as the default value for the lower half-word of the \texttt{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 Scanlines per Frame**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the 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 Pixels Per Scanline**: 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 Pixels per Scan line** field as for constant mode the maximum number of pixels equals the active number of pixels.

---

**Output Generation**

For details, see “Generating IP Output Products” in the *Vivado Design Suite User Guide: Designing with IP* (UG896).
Constraining the Core

Required Constraints

The only constraints required are clock frequency constraints for the video clock, \( \text{clk} \), and the AXI4-Lite clock, \( \text{s_axi_aclk} \). Paths between the two clock domains should be constrained with a \text{max\_delay} constraint and use the \text{datapathonly} flag, causing setup and hold checks to be ignored for signals that cross clock domains. These constraints are provided in the XDC constraints file included with the core.
Simulation

This chapter contains information about simulating IP in the Vivado® Design Suite environment. For comprehensive information about Vivado simulation components, as well as information about using supported third party tools, see the Vivado Design Suite User Guide: Logic Simulation (UG900) [Ref 6].
Synthesis and Implementation

For details about synthesis and implementation, see “Synthesizing IP” and “Implementing IP” in the Vivado Design Suite User Guide: Designing with IP (UG896) [Ref 3].
C Model Reference

The Image Enhancement core has a bit accurate C model designed for system modeling.

Features

- Bit-accurate with the Image Enhancement v8.0 core
- Statically linked library (.lib for Windows)
- Dynamically linked library (.so for Linux)
- Available for 32-bit and 64-bit Windows platforms and 32-bit and 64-bit Linux platforms
- Supports all features of the Image Enhancement core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C code showing how to use the function is provided
- Example application C code wrapper file supports 8-bit YUV and BIN

Overview

The Image Enhancement core has a bit-accurate C model for 32-bit and 64-bit Windows platforms and 32-bit and 64-bit Linux platforms. The model's interface consists of a set of C functions residing in a statically linked library (shared library).

See Using the C Model for full details of the interface. A C code example of how to call the model is provided in C Model Example Code.

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, and it does not model the core's latency or its interface signals.
Unpacking and Model Contents

Unzip the `v_enhance_v8_0_bitacc_cmodel_<OS>.zip` file, containing the bit accurate models for the Image Enhancement IP Core. This creates the files in Table 8-1.

**Table 8-1: Bit Accurate C Model Directory Structure and Files**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>v_enhance_v8_0_bitacc_cmodel.h</code></td>
<td>Model header file</td>
</tr>
<tr>
<td><code>rgb_utils.h</code></td>
<td>Header file declaring the RGB image/video container type and support functions</td>
</tr>
<tr>
<td><code>yuv_utils.h</code></td>
<td>Header file declaring the YUV (.yuv) image file I/O functions</td>
</tr>
<tr>
<td><code>video_utils.h</code></td>
<td>Header file declaring the generalized image/video container type, I/O and support functions</td>
</tr>
<tr>
<td><code>parsers.h</code></td>
<td>Header file for configuration file parser</td>
</tr>
<tr>
<td><code>run_bitacc_cmodel.c</code></td>
<td>Example code calling the C model</td>
</tr>
<tr>
<td><code>parsers.c</code></td>
<td>Code for reading configuration file file</td>
</tr>
<tr>
<td><code>/examples</code></td>
<td>Example input files used by C model</td>
</tr>
<tr>
<td><code>enhance_example.cfg</code></td>
<td>Sample configuration file containing the core parameter settings</td>
</tr>
<tr>
<td><code>input_image.yuv</code></td>
<td>Sample test image</td>
</tr>
<tr>
<td><code>input_image.hdr</code></td>
<td>Sample test image header file</td>
</tr>
<tr>
<td>files included in the lin64.zip file</td>
<td>Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms</td>
</tr>
<tr>
<td><code>libIp_v_enhance_v8_0_bitacc_cmodel.so</code></td>
<td>Model shared object library</td>
</tr>
<tr>
<td>files included in the lin.zip file</td>
<td>Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms</td>
</tr>
<tr>
<td><code>libIp_v_enhance_v8_0_bitacc_cmodel.so</code></td>
<td>Model shared object library</td>
</tr>
<tr>
<td>files included in the nt.zip file</td>
<td>Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms</td>
</tr>
<tr>
<td><code>libIp_v_enhance_v8_0_bitacc_cmodel.dll</code></td>
<td>Precompiled library file for win32 compilation</td>
</tr>
<tr>
<td><code>libIp_v_enhance_v8_0_bitacc_cmodel.lib</code></td>
<td>Precompiled library file for win64 compilation</td>
</tr>
<tr>
<td>files included in the nt64.zip file</td>
<td>Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms</td>
</tr>
<tr>
<td><code>libIp_v_enhance_v8_0_bitacc_cmodel.dll</code></td>
<td>Precompiled library file for win64 compilation</td>
</tr>
<tr>
<td><code>libIp_v_enhance_v8_0_bitacc_cmodel.lib</code></td>
<td>Precompiled library file for win64 compilation</td>
</tr>
</tbody>
</table>
Installation

For Linux, make sure this file is in a directory that is in your $LD_LIBRARY_PATH environment variable:

- libIp_v_enhance_v8_0_bitacc_cmodel.so

Software Requirements

The Image Enhancement v8.0 C models were compiled and tested with the software listed in Table 8-2.

Table 8-2: Compilation Tools for the Bit Accurate C Models

<table>
<thead>
<tr>
<th>Platform</th>
<th>C Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>32- and 64-bit Linux</td>
<td>GCC 4.1.1</td>
</tr>
<tr>
<td>32- and 64-bit Windows</td>
<td>Microsoft Visual Studio 2008</td>
</tr>
</tbody>
</table>

Using the C Model

The bit accurate C model is accessed through a set of functions and data structures that are declared in the v_enhance_v8_0_bitacc_cmodel.h file.

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

```c
struct xilinx_ip_v_enhance_v8_0_generics enhance_generics;
struct xilinx_ip_v_enhance_v8_0_inputs   enhance_inputs;
struct xilinx_ip_v_enhance_v8_0_outputs  enhance_outputs;
```

The declaration of these structures is in the v_enhance_v8_0_bitacc_cmodel.h file.

Table 8-3 lists the generic parameters taken by the Image Enhancement v8.0 IP core bit accurate model, as well as the default values. For an actual instance of the core, these parameters can only be set in generation time through the GUI.

Table 8-3: C Model Generic Parameters and Default Values

<table>
<thead>
<tr>
<th>Generic Variable</th>
<th>Type</th>
<th>Default Value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_AXIS_VIDEO_DATA_WIDTH</td>
<td>int</td>
<td>8</td>
<td>8,10,12,16</td>
<td>Data width</td>
</tr>
</tbody>
</table>
Using the C Model

Calling `xilinx_ip_v_enhance_v8_0_get_default_generics(&enhance_generics)` initializes the generics structure with the Image Enhancement GUI defaults, listed in Table 8-3:

Enhancement parameters can also be set dynamically through the AXI4-Lite interface. Consequently, these values are passed as inputs to the core, along with the actual test image, or video sequence (Table 8-4).

### Table 8-3: C Model Generic Parameters and Default Values (Cont’d)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default Value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAS_NOISE</td>
<td>int</td>
<td>1, 0</td>
<td>1=perform noise reduction, 0=no noise reduction</td>
<td></td>
</tr>
<tr>
<td>HAS_ENHANCE</td>
<td>int</td>
<td>1, 0</td>
<td>1=perform edge enhancement, 0=no edge enhancement</td>
<td></td>
</tr>
<tr>
<td>OPT_SIZE</td>
<td>int</td>
<td>0, 1</td>
<td>1=Minimal Resources algorithm, 0=High Performance algorithm</td>
<td></td>
</tr>
<tr>
<td>HAS_HALO</td>
<td>int</td>
<td>1, 0</td>
<td>1=add halo suppression, 0=no halo suppression</td>
<td></td>
</tr>
<tr>
<td>HAS_ALIAS</td>
<td>int</td>
<td>0, 1</td>
<td>1=add anti-aliasing filtering, 0=no anti-alias filtering</td>
<td></td>
</tr>
</tbody>
</table>

### Table 8-4: Core Generic Parameters and Default Values

<table>
<thead>
<tr>
<th>Input Variable</th>
<th>Type</th>
<th>Default Value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>video_in</td>
<td>video_struct</td>
<td>null</td>
<td>N/A</td>
<td>Container to hold input image or video data.</td>
</tr>
<tr>
<td>noise_threshold</td>
<td>int</td>
<td>192</td>
<td>Allowed values are 0 to $2^{\text{C_AXIS_VIDEO_DATA_WIDTH}}-1$</td>
<td>Amount of noise reduction applied by the low-pass filters. An image with more noise will need a higher NOISE_THRESHOLD value. A value of 0 will apply no filtering. A value of 1 will apply filtering to the entire image (with weighting based on the Edge Map).</td>
</tr>
</tbody>
</table>
The structure `enhance_inputs` defines the values of run time parameters and the actual input image. Calling `xilinx_ip_v_enhance_v8_0_get_default_inputs(&enhance_generics, &enhance_inputs)` initializes the input structure with the default values (see Table 8-4).

**IMPORTANT:** *The video_in variable is not initialized because the initialization depends on the actual test image to be simulated. Chapter 8, C Model Example Code describes the initialization of the video_in structure.*

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

```c
int xilinx_ip_v_enhance_v8_0_bitacc_simulate(
    struct xilinx_ip_v_enhance_v8_0_generics* generics,
    struct xilinx_ip_v_enhance_v8_0_inputs* inputs,
    struct xilinx_ip_v_enhance_v8_0_outputs* outputs).
```

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

```c
void xilinx_ip_v_enhance_v8_0_destroy(
    struct xilinx_ip_v_enhance_v8_0_inputs *input,
    struct xilinx_ip_v_enhance_v8_0_outputs *output).
```
Successful execution of all provided functions, except for the destroy function, return value 0. A non-zero error code indicates that problems occurred during function calls.

Image Enhancement Input and Output Video Structure

Input images or video streams can be provided to the Image Enhancement v8.0 reference model using the `video_struct` structure, defined in `video_utils.h`:

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

Table 8-5: Member Variables of the Video Structure

<table>
<thead>
<tr>
<th>Member Variable</th>
<th>Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>frames</td>
<td>Number of video/image frames in the data structure.</td>
</tr>
<tr>
<td>rows</td>
<td>Number of rows per frame.</td>
</tr>
<tr>
<td>cols</td>
<td>Number of columns per frame.</td>
</tr>
<tr>
<td>bits_per_component</td>
<td>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.</td>
</tr>
<tr>
<td>mode</td>
<td>Contains information about the designation of data planes.</td>
</tr>
<tr>
<td>data</td>
<td>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].</td>
</tr>
</tbody>
</table>

Table 8-6: Named Video Modes with Corresponding Planes and Representations

<table>
<thead>
<tr>
<th>Mode</th>
<th>Planes</th>
<th>Video Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT_MONO</td>
<td>1</td>
<td>Monochrome – Luminance only</td>
</tr>
<tr>
<td>FORMAT_RGB</td>
<td>3</td>
<td>RGB image/video data</td>
</tr>
<tr>
<td>FORMAT_C444</td>
<td>3</td>
<td>444 YUV, or YCrCb image/video data</td>
</tr>
<tr>
<td>FORMAT_C422</td>
<td>3</td>
<td>422 format YUV video, (u, v chrominance channels horizontally sub-sampled)</td>
</tr>
<tr>
<td>FORMAT_C420</td>
<td>3</td>
<td>420 format YUV video, (u, v sub-sampled both horizontally and vertically)</td>
</tr>
</tbody>
</table>
Using the C Model

The Image Enhancement core supports the mode FORMAT_C444 and FORMAT_C422.

### Initializing the Image Enhancement Input Video Structure

The easiest way to assign stimuli values to the input video structure is to initialize it with an image or video. The `yuv_utils.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`.

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

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

```c
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.

```c
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`.

```c
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 8-6. 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:

```c
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
```

The structure of .bin files are described in Binary Image/Video Files.
To ease modifying and debugging the provided top-level demonstrator using the built-in debugging environment of Visual Studio, the top-level command line parameters can be specified through the Project Property Pages using these steps:

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

### Compiling Image Enhancement C Model with Example Wrapper

#### Linux (32-bit and 64-bit)

To compile the example code, perform these steps:

1. Set your $LD_LIBRARY_PATH environment variable to include the root directory where you unzipped the model zip file using a command such as:
   ```
   setenv LD_LIBRARY_PATH <unzipped_c_model_dir>:${LD_LIBRARY_PATH}
   ```
2. Copy this file from the /lin64 directory to the root directory:
   ```
   libIp_v_enhance_v8_0_bitacc_cmodel.so
   ```
3. In the root directory, compile using the GNU C Compiler with this command:
   ```
   gcc -m32 -x c++ ../run_bitacc_cmodel.c ../gen_stim.c ../parsers.c -o run_bitacc_cmodel -L. -lIp_v_enhance_v8_0_bitacc_cmodel -Wl,-rpath,.
   ```
   ```
   gcc -m64 -x c++ ../run_bitacc_cmodel.c ../gen_stim.c ../parsers.c -o run_bitacc_cmodel -L. -lIp_v_enhance_v8_0_bitacc_cmodel -Wl,-rpath,.
   ```

#### Windows (32-bit and 64-bit)

The precompiled library `v_enhance_v8_0_bitacc_cmodel.lib`, and top-level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. An example procedure is provided here using Microsoft Visual Studio.

1. In Visual Studio, create a new, empty Win32 Console Application project.
2. As existing items, add:
   a. `libIp_v_enhance_v8_0_bitacc_cmodel.lib` to the Resource Files folder of the project
b. run_bitacc_cmodel.c, parsers.c, and gen_stim.c to the Source Files folder of the project

c. v_enhance_v8_0_bitacc_cmodel.h and parsers.h to the Header Files folder of the project

3. After the project is created and populated, it must be compiled and linked (built) to create a win32/win64 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.
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.
Test Bench

This chapter contains information about the provided test bench in the Vivado® Design Suite environment.

Demonstration Test Bench

A demonstration test bench is provided with the core which enables you to observe core behavior in a typical scenario. This test bench is generated together with the core in Vivado Design Suite. You are encouraged to make simple modifications to the configurations and observe the changes in the waveform.

Directory and File Contents

The following files are expected to be generated in the demonstration test bench output directory:

- axi4lite_mst.v
- axi4s_video_mst.v
- axi4s_video_slv.v
- ce_generator.v
- tb_<IP_instance_name>.v

Test Bench Structure

The top-level entity is \texttt{tb\_<IP\_instance\_name>}. It instantiates the following modules:

- \texttt{DUT}
  - The \texttt{<IP> core instance under test.}
- \texttt{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 generates ramp data and initiates AXI4-Stream transactions to provide video stimuli for the core and can also be used to open stimuli files generated from the reference C models and convert them into corresponding AXI4-Stream transactions.

  To do this, edit `tb_<IP_instance_name>.v`:
  
  a. Add define macro for the stimuli file name and directory path
     
   ```
   define STIMULI_FILE_NAME <path><filename>.
   ```

  b. Comment-out/remove the following line:

   ```
   MST.is_ramp_gen(`C_ACTIVE_ROWS, `C_ACTIVE_COLS, 2);
   ```

   and replace with the following line:

   ```
   MST.use_file(`STIMULI_FILE_NAME);
   ```

  For information on how to generate stimuli files, see Chapter 4, C Model Reference.

- **axi4s_video_slv**

  The AXI4-Stream slave module, which acts as a passive slave to provide handshake signals for the AXI4-Stream transactions from the core output, can be used to open the data files generated from the reference C model and verify the output from the core.

  To do this, edit `tb_<IP_instance_name>.v`:

  a. Add define macro for the golden file name and directory path
     
   ```
   define GOLDEN_FILE_NAME "<path><filename>".
   ```

  b. Comment out the following line:

   ```
   SLV.is_passive;
   ```

   and replace with the following line:

   ```
   SLV.use_file(`GOLDEN_FILE_NAME);
   ```

  For information on how to generate golden files, see Chapter 4, C Model Reference.

- **ce_gen**

  Programmable Clock Enable (ACLKEN) generator.
Appendix A

Verification, Compliance, and Interoperability

Simulation
A highly parameterizable test bench was used to test the Image Enhancement 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 Enhancement 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 Enhancement core
  - Launching the test
  - Comparing the output of the core against the expected results
Interoperability

The core slave (input) AXI4 Stream interface can work directly with any Xilinx Video core that produces YCbCr 4:4:4 or 4:2:2. The core master (output) interface can work directly with any Xilinx Video core which consumes YCbCr 4:4:4 or 4:2:2 data.

The AXI4-Stream interfaces must be compliant to the AXI4-Stream Video Protocol as described in Video IP: AXI Feature Adoption section of the AXI Reference Guide [Ref 1].
Appendix B

Migrating and Upgrading

This appendix contains information about migrating from an ISE design to the Vivado Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading their IP core, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information about migration to Vivado Design Suite, see ISE to Vivado Design Suite Migration Guide (UG911) [Ref 2].

Upgrading in Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Parameter Changes

The Image Enhancement v8.0 core supports the functionality of both the Image Edge Enhancement v7.0 and Image Noise Reduction v6.0 cores.

From version v.7.0 of the Image Enhancement core and v6.0 of the Image Noise Reduction core to v8.0 of the Image Enhancement core, the following significant changes took place:

- New algorithm for image edge enhancement with higher quality results
  - Removed gain parameters from previous versions of the core
  - New parameter for controlling amount of edge enhancement
  - New halo suppression feature added
  - New anti-aliasing feature added
- Support added for image noise reduction

Discontinued IP
Both noise reduction and edge enhancement are executed in a single core (there is no need for a separate Image Noise Reduction core)

- New algorithm for image noise reduction with higher quality results
- Removed strength parameter from previous Image Noise Reduction core
- New parameter for controlling amount of noise reduction

- Two algorithms available when executing both noise reduction and edge enhancement: Minimal Resources algorithm and High Performance algorithm

The Vivado Upgrade IP function will upgrade from previous versions of the Image Edge Enhancement core and set all the Image Enhancement v8.0 parameters to the default values. The parameters must be re-configured for the user’s specific application.

**Upgrading from Image Edge Enhancement Core**

The new v8.0 Enhance Strength parameter does not map directly to the former core’s Gain parameters. You evaluate and set the Enhance Strength value needed for the your specific application. Also, optional post-processing modules for anti-halo and anti-aliasing are available for reducing any artifacts introduced by the high-pass filtering of the edge enhancement algorithm. See Enhance Strength in Chapter 3 for more information.

**Upgrading from Image Noise Reduction Core**

The Image Enhancement v8.0 core supersedes the Image Noise Reduction core. The Image Noise Reduction core in your design can be replaced with the Image Edge Enhancement v8.0 core. A new algorithm is used for image noise reduction, and the new Noise Threshold parameter does not map directly to the former core’s Filter Strength parameter. You must evaluate and set the Noise Threshold value needed for your specific application. See Noise Threshold in Chapter 3 for more information.

**Upgrading Designs Containing Both Image Noise Reduction and Image Edge Enhancement Cores**

The Image Enhancement v8.0 core combines the functionality of both the Image Edge Enhancement and Image Noise Reduction cores. There is no longer a need for two separate cores. If the Image Noise Reduction core and Image Edge Enhancement core were used in sequence, then the Image Noise Reduction core can be removed from the design. The Image Enhancement v8.0 core can be parameterized to perform both noise reduction and edge enhancement. You must evaluate and set all the core parameter values for your specific application.

**Port Changes**

There are no port changes.
Other Changes

From version v.7.0 of the Image Enhancement core and v6.0 of the Image Noise Reduction core to v8.0 of the Image Enhancement core, the following significant changes took place:

• Support for YCbCr 4:4:4 and 4:2:2.
• Support for 16 bit Video Component Width (in addition to 8, 10, and 12).
• New version of software driver.
• Added support for C simulation model delivery via Vivado.
Appendix C

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the Image Enhancement, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support Web Case.

Documentation

This product guide is the main document associated with the Image Enhancement. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are listed below, and can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered
A filter search is available after results are returned to further target the results.

**Answer Records for the Image Enhancement Core**

AR 54525

**Technical Support**

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

To contact Xilinx Technical Support, navigate to the [Xilinx Support web page](https://www.xilinx.com).

1. Open a WebCase by selecting the [WebCase](https://www.xilinx.com) link located under Support Quick Links.
   - A block diagram of the video system that explains the video source, destination and IP (custom and Xilinx) used.

*Note:* Access to WebCase is not available in all cases. Please login to the WebCase tool to see your specific support options.

---

**Debug Tools**

There are many tools available to address Image Enhancement core design issues. It is important to know which tools are useful for debugging various situations.

**Vivado Lab Tools**

Vivado® lab tools insert logic analyzer and virtual I/O cores directly into your design. Vivado lab tools allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature represents the functionality in the Vivado IDE that is used for logic debugging and validation of a design running in Xilinx devices in hardware.

The Vivado lab tools logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:
Appendix C: Debugging

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)


### Reference Boards

Various Xilinx development boards support Image Enhancement. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series evaluation boards
  - KC705
  - KC724

### C Model Reference

See *C Model Reference* in this guide for tips and instructions for using the provided C model files to debug your design.

---

### Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado lab tools are a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado lab tools for debugging the specific problems.

### General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the *LOCKED* port.
- If your outputs go to 0, check your licensing.
**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 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, Green, Blue, Cyan, Red, Yellow, Magenta, 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 monitoring 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 multiport 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 multiport 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 datapaths.

The \texttt{SYSDEBUG0} (0x0014) (or Frame Throughput Monitor) indicates the number of frames processed since power-up or the last time the core was reset. The \texttt{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 \texttt{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 Enhancement hardware evaluation core times out after approximately eight hours of operation. The output is driven to zero. This results in a black screen for RGB color systems and in a dark-green screen for YUV color systems.

---

**Interface Debug**

**AXI4-Lite Interfaces**

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

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readback from the Version Register through the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.</td>
<td>Are the \texttt{S_AXI_ACLK} and \texttt{ACLK} pins connected? The \texttt{VERSION REGISTER} readout issue may be indicative of the core not receiving the AXI4-Lite interface.</td>
</tr>
<tr>
<td>Readback from the Version Register through the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.</td>
<td>Is the core enabled? Is \texttt{s_axi_aclken} connected to vcc? Verify that signal \texttt{ACLKEN} is connected to either \texttt{net_vcc} or to a designated clock enable signal.</td>
</tr>
</tbody>
</table>
Table C-1: Troubleshooting the AXI4-Lite Interface (Cont’d)

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readback from the Version Register through the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.</td>
<td>Is the core in reset? S_AXIS_ARESETn and ARESETn should be connected to vcc for the core not to be in reset. Verify that the S_AXIS_ARESETn and ARESETn signals are connected to either net_vcc or to a designated reset signal.</td>
</tr>
<tr>
<td>Readback value for the VERSION_REGISTER is different from expected default values</td>
<td>The core and/or the driver in a legacy project has not been updated. Ensure that old core versions, implementation files, and implementation caches have been cleared.</td>
</tr>
</tbody>
</table>

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

**AXI4-Stream Interfaces**

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

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

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0 of the ERROR register reads back set.</td>
<td>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 Vivado Lab Tools, measure the number of active AXI4-Stream transactions between EOL pulses.</td>
</tr>
<tr>
<td>Bit 1 of the ERROR register reads back set.</td>
<td>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 Vivado Lab Tools, measure the number of active AXI4-Stream transactions between EOL pulses.</td>
</tr>
<tr>
<td>Bit 2 or Bit 3 of the ERROR register reads back set.</td>
<td>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 Vivado Lab Tools, measure the number EOL pulses between subsequent SOF pulses.</td>
</tr>
<tr>
<td>s_axis_video_tready stuck low, the upstream core cannot send data.</td>
<td>During initialization, line-, and frame-flushing, the 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 core cannot send data downstream, and the internal FIFOs are full.</td>
</tr>
</tbody>
</table>
Appendix C: Debugging

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

**Other Interfaces**

*Table C-3* describes how to troubleshoot third-party interfaces.

**Table C-2:** Troubleshooting AXI4-Stream Interface *(Cont’d)*

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Solution</th>
</tr>
</thead>
</table>
| m_axis_video_tvalid stuck low, the downstream core is not receiving data | - No data is generated during the first two lines of processing.  
- 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 this core. Inconsistent EOL and/or SOF periods received. | - Are the Master and Slave AXI4-Stream interfaces in the same clock domain?  
- Is proper clock-domain crossing logic instantiated between the upstream core and this core (Asynchronous FIFO)?  
- Did the design meet timing?  
- Is the frequency of the clock source driving the ACLK pin lower than the reported Fmax reached? |
| Data samples lost between Downstream core and this core. Inconsistent EOL and/or SOF periods received. | - Are the Master and Slave AXI4-Stream interfaces in the same clock domain?  
- Is proper clock-domain crossing logic instantiated between the upstream core and this core (Asynchronous FIFO)?  
- Did the design meet timing?  
- Is the frequency of the clock source driving the ACLK pin lower than the reported Fmax reached? |
Table C-3: Troubleshooting Third-Party Interfaces

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Severe color distortion or color-swap when interfacing to third-party video IP.</td>
<td>Verify that the color component logical addressing on the AXI4-Stream TDATA signal is in according to Data Interface in Chapter 2. If misaligned: In HDL, break up the TDATA vector to constituent components and manually connect the slave and master interface sides.</td>
</tr>
<tr>
<td>Severe color distortion or color-swap when processing video written to external memory using the AXI-VDMA core.</td>
<td>Unless the particular software driver was developed with the AXI4-Stream TDATA signal color component assignments described in Data Interface in Chapter 2 in mind, there are no guarantees that the software correctly identifies bits corresponding to color components. Verify that the color component logical addressing TDATA is in alignment with the data format expected by the software drivers reading/writing external memory. If misaligned: In HDL, break up the TDATA vector to constituent components, and manually connect the slave and master interface sides.</td>
</tr>
</tbody>
</table>
Appendix D

Additional Resources

Xilinx Resources

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

Xilinx Support web page.

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


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:


References

These documents provide supplemental material useful with this user guide:

1. Vivado AXI Reference Guide (UG1037)
2. ISE to Vivado Design Suite Migration Guide (UG911)
## Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/18/2015</td>
<td>8.0</td>
<td>Added UltraScale+ support.</td>
</tr>
<tr>
<td>10/01/2014</td>
<td>8.0</td>
<td>Removed Application Software Development appendix.</td>
</tr>
<tr>
<td>12/18/2013</td>
<td>8.0</td>
<td>Added UltraScale Architecture support.</td>
</tr>
<tr>
<td>10/02/2013</td>
<td>8.0</td>
<td>Updated Constraints and Test Bench chapters.</td>
</tr>
<tr>
<td>06/19/2013</td>
<td>8.0</td>
<td>Synchronized document version with core version. Updates for new</td>
</tr>
<tr>
<td></td>
<td></td>
<td>algorithm for image edge enhancement and new support added for image</td>
</tr>
<tr>
<td></td>
<td></td>
<td>noise reduction.</td>
</tr>
<tr>
<td>03/20/2013</td>
<td>4.0</td>
<td>Updated for core version. Removed ISE chapters.</td>
</tr>
<tr>
<td>12/18/2012</td>
<td>3.2</td>
<td>Updated for core version and ISE 14.4 and Vivado 2012.4 design tools.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated resource numbers, Debugging appendix, and gain registers.</td>
</tr>
<tr>
<td>10/16/2012</td>
<td>3.1</td>
<td>Updated for core version and ISE 14.3 and Vivado 2012.3 design tools.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Vivado test bench.</td>
</tr>
<tr>
<td>07/25/2012</td>
<td>3.0</td>
<td>Updated for core version. Added Vivado information.</td>
</tr>
<tr>
<td>04/24/2012</td>
<td>2.0</td>
<td>Updated for core version. Added Zynq-7000 devices, added AXI4-Stream</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interfaces, deprecated GPP interface.</td>
</tr>
<tr>
<td>10/19/2011</td>
<td>1.0</td>
<td>Initial Xilinx release of Product Guide, replacing DS753 and UG831.</td>
</tr>
</tbody>
</table>

## 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](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](http://www.xilinx.com/warranty.htm#critapps).

© Copyright 2011–2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, 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.