# Table of Contents

## SECTION I: SUMMARY

### IP Facts

#### Chapter 1: Overview

- Overview ............................................................................................................. 7
- Feature Summary .................................................................................................. 8
- Applications ......................................................................................................... 8
- Licensing and Ordering Information ..................................................................... 8

#### Chapter 2: Product Specification

- Standards .............................................................................................................. 9
- Performance ......................................................................................................... 9
- Resource Utilization ............................................................................................... 10
- Core Interfaces and Register Space ...................................................................... 12

#### Chapter 3: Designing with the Core

- General Design Guidelines .................................................................................. 27
- Clock, Enable, and Reset Considerations ............................................................. 28
- System Considerations .......................................................................................... 30

#### Chapter 4: C Model Reference

- Features ............................................................................................................... 32
- Overview .............................................................................................................. 32
- Unpacking and Model Contents .......................................................................... 33
- Installation ........................................................................................................... 34
- Software Requirements ....................................................................................... 34
- Using the C Model ............................................................................................... 34
- C Model Example Code ....................................................................................... 39

## SECTION II: VIVADO DESIGN SUITE
Chapter 5: Customizing and Generating the Core
Graphical User Interface ................................................................. 42

Chapter 6: Constraining the Core
Required Constraints ................................................................. 45
Device, Package, and Speed Grade Selections ................................ 45
Clock Frequencies ................................................................. 45
Clock Management .............................................................. 46
Clock Placement ................................................................. 46
Banking ................................................................. 46
Transceiver Placement .......................................................... 46
I/O Standard and Placement ................................................ 46

Chapter 7: Detailed Example Design
Demonstration Test Bench .......................................................... 47

SECTION III: ISE DESIGN SUITE

Chapter 8: Customizing and Generating the Core
Graphical User Interface ................................................................. 54
Parameter Values in the XCO File .................................................. 56
Output Generation ................................................................. 57

Chapter 9: Constraining the Core
Required Constraints ................................................................. 59
Device, Package, and Speed Grade Selections ................................ 59
Clock Frequencies ................................................................. 59
Clock Management .............................................................. 60
Clock Placement ................................................................. 60
Banking ................................................................. 60
Transceiver Placement .......................................................... 60
I/O Standard and Placement ................................................ 60

Chapter 10: Detailed Example Design
Demonstration Test Bench .......................................................... 61
Test Bench Structure ............................................................. 61
Running the Simulation .......................................................... 62
Directory and File Contents .................................................. 62
SECTION IV: APPENDICES

Appendix A: Verification, Compliance, and Interoperability
Simulation ................................................................................................................. 65
Hardware Testing ...................................................................................................... 65
Interoperability ........................................................................................................ 66

Appendix B: Migrating

Appendix C: Debugging
Finding Help on Xilinx.com ..................................................................................... 68
Debug Tools ............................................................................................................... 70
Hardware Debug ...................................................................................................... 71
Interface Debug ...................................................................................................... 73

Appendix D: Application Software Development
Programmer Guide .................................................................................................. 77

Appendix E: Additional Resources
Xilinx Resources ..................................................................................................... 80
Solution Centers ........................................................................................................ 80
References .............................................................................................................. 80
Technical Support .................................................................................................... 81
Revision History ....................................................................................................... 81
Notice of Disclaimer ................................................................................................. 81
SECTION I: SUMMARY

IP Facts
Overview
Product Specification
Designing with the Core
C Model Reference
Introduction

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

Features

• Programmable gain for edge directions
• YCbCr 4:4:4 input and output
• AXI4-Stream data interfaces
• Optional AXI4-Lite control interface
• Supports 8, 10, and 12-bits per color component input and output
• Built-in, optional bypass and test-pattern generator mode
• Built-in, optional throughput monitors
• Supports spatial resolutions from 32x32 up to 7680x7680
  • Supports 1080P60 in all supported device families (1)
  • Supports 4kx2k @ 24 Hz in supported high performance devices

1. Performance on low power devices may be lower.

---

<table>
<thead>
<tr>
<th>LogiCORE IP Facts Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core Specifics</strong></td>
</tr>
<tr>
<td>Supported Device Family(1)</td>
</tr>
<tr>
<td>Supported User Interfaces</td>
</tr>
<tr>
<td>Resources</td>
</tr>
<tr>
<td><strong>Provided with Core</strong></td>
</tr>
<tr>
<td>Documentation</td>
</tr>
<tr>
<td>Design Files</td>
</tr>
<tr>
<td>Example Design</td>
</tr>
<tr>
<td>Test Bench</td>
</tr>
<tr>
<td>Constraints File</td>
</tr>
<tr>
<td>Simulation Models</td>
</tr>
<tr>
<td>Supported Software Drivers</td>
</tr>
<tr>
<td><strong>Tested Design Flows</strong></td>
</tr>
<tr>
<td>Design Entry Tools</td>
</tr>
<tr>
<td>Simulation(5)</td>
</tr>
<tr>
<td>Synthesis Tools</td>
</tr>
<tr>
<td><strong>Support</strong></td>
</tr>
<tr>
<td>Provided by Xilinx, Inc.</td>
</tr>
</tbody>
</table>

1. For a complete listing of supported devices, see the release notes for this core.
2. Supported in ISE Design Suite implementations only.
3. Video protocol as defined in the Video IP: AXI Feature Adoption section of the (UG761) AXI Reference Guide [Ref 1].
5. For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.
6. Supports only 7 series devices.
Overview

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

Figure 1-1: Image Edge Enhancement
Feature Summary

The Image Edge Enhancement core uses Sobel and Laplacian filters to enhance edges of objects. There is a programmable gain for each filter to adjust the strength of the edge enhancement effect. This core works on YCbCr 4:4:4 data. The core is capable of a maximum resolution of 7680 columns by 7680 rows with 8, 10, or 12 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 CORE Generator or EDK 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/ISE 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 Edge Enhancement product web page.

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 Edge Enhancement core is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. Refer to the Video IP: AXI Feature Adoption section of the (UG761) AXI Reference Guide [Ref 1] for additional information.

Performance

The following sections detail the performance characteristics of the Image Edge 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 FPGA device, using a different version of Xilinx tools and other factors. Refer to in Table 2-1 through Table 2-8 for device-specific information.

Latency

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

Throughput

The Image Edge Enhancement core 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 19 clock cycles, during which the s_axis_video_tready is de-asserted signaling that the core is not ready to process samples. Also at the end of each frame the core flushes internal line buffers for 1 scan line, during which the s_axis_video_tready is de-asserted signaling that the core is not ready to process samples.

When the core is processing timed streaming video (which is typical for 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, e.g. a frame buffer, the throughput of the core can be defined as follows:

\[
R_{MAX} = f_{ACLK} \times \frac{ROWS}{ROWS + 1} \times \frac{COLS}{COLS + 19}
\]

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

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

\[
f_{ACLK} = \frac{R_{MAX} \times \frac{ROWS + 1}{ROWS} \times \frac{COLS + 19}{COLS}}{124.4 \times \frac{1081}{1080} \times \frac{1939}{1920} = 125.4}
\]

### 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.

### Resource Utilization using Vivado Design Suite

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 Edge Enhancement core for all input/output width combinations for Virtex-7, Kintex-7, Artix-7 FPGA families using the Vivado Design Suite. This core does not use any dedicated I/O or CLK resources. The design was tested with the AXI4-Lite interface, INTC_IF and the Debug Features disabled. By default, the maximum number of pixels per scan line was set to 1920, active pixels per scan line was set to 1920.
Resource Utilization using ISE Design Suite

The information presented in Table 2-4 through Table 2-8 is a guide to the resource utilization and maximum clock frequency of the Image Edge Enhancement core for all input/output width combinations for Virtex-7, Kintex-7, Artix-7, Zynq-7000, Virtex-6, and Spartan-6 FPGA families. The design was tested using ISE® v14.4 tools with default tool options for characterization data. This core does not use any dedicated I/O or CLK resources. The design was tested with the AXI4-Lite interface, INTC_IF and the Debug Features disabled. By default, the maximum number of pixels per scan line was set to 1920, active pixels per scan line was set to 1920.

Table 2-1:  Kintex-7

<table>
<thead>
<tr>
<th>Data Width</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAM 36 / 18</th>
<th>DSP48S</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1087</td>
<td>806</td>
<td>1063</td>
<td>2/1</td>
<td>1</td>
<td>274</td>
</tr>
<tr>
<td>10</td>
<td>1196</td>
<td>922</td>
<td>1249</td>
<td>5/0</td>
<td>1</td>
<td>266</td>
</tr>
<tr>
<td>12</td>
<td>1439</td>
<td>1081</td>
<td>1435</td>
<td>12/0</td>
<td>1</td>
<td>274</td>
</tr>
</tbody>
</table>

Device, Part, Speed: XC7K70T,FBG484,-1 (ADVANCED 1.08a 2012-11-02)

Table 2-2:  Artix-7

<table>
<thead>
<tr>
<th>Data Width</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAM 36 / 18</th>
<th>DSP48S</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1089</td>
<td>804</td>
<td>1063</td>
<td>2/1</td>
<td>1</td>
<td>212</td>
</tr>
<tr>
<td>10</td>
<td>1259</td>
<td>920</td>
<td>1249</td>
<td>5/0</td>
<td>1</td>
<td>219</td>
</tr>
<tr>
<td>12</td>
<td>1448</td>
<td>1082</td>
<td>1435</td>
<td>12/0</td>
<td>1</td>
<td>196</td>
</tr>
</tbody>
</table>

Device, Part, Speed: XC7A100T,FGG484,-1 (ADVANCED 1.06c 2012-11-02)

Table 2-3:  Virtex-7

<table>
<thead>
<tr>
<th>Data Width</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAM 36 / 18</th>
<th>DSP48S</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1111</td>
<td>806</td>
<td>1063</td>
<td>2/1</td>
<td>1</td>
<td>266</td>
</tr>
<tr>
<td>10</td>
<td>1280</td>
<td>921</td>
<td>1249</td>
<td>5/0</td>
<td>1</td>
<td>274</td>
</tr>
<tr>
<td>12</td>
<td>1377</td>
<td>1080</td>
<td>1435</td>
<td>12/0</td>
<td>1</td>
<td>281</td>
</tr>
</tbody>
</table>

Device, Part, Speed: XC7V585T,FFG1157,-1 (PRODUCTION 1.08b 2012-11-02)

Table 2-4:  Artix-7 and Zynq-7000 Devices with Artix Based Fabric

<table>
<thead>
<tr>
<th>Data Width</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAM 36 / 18</th>
<th>DSP48E1</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1109</td>
<td>967</td>
<td>1019</td>
<td>2/1</td>
<td>1</td>
<td>212</td>
</tr>
<tr>
<td>10</td>
<td>1302</td>
<td>1095</td>
<td>1201</td>
<td>5/0</td>
<td>1</td>
<td>196</td>
</tr>
<tr>
<td>12</td>
<td>1478</td>
<td>1242</td>
<td>1383</td>
<td>12/0</td>
<td>1</td>
<td>204</td>
</tr>
</tbody>
</table>

Device, Part, Speed: XC7A100T,FGG484,-1 (ADVANCED 1.06d 2012-11-12)
Core Interfaces and Register Space

Port Descriptions

The Image Edge 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 Edge 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.

### Table 2-5: Kintex-7 and Zynq-7000 Devices with Kintex Based Fabric

<table>
<thead>
<tr>
<th>Data Width</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAM 36 / 18</th>
<th>DSP48E1</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1106</td>
<td>985</td>
<td>1019</td>
<td>2/1</td>
<td>1</td>
<td>288</td>
</tr>
<tr>
<td>10</td>
<td>1260</td>
<td>1116</td>
<td>1201</td>
<td>5/0</td>
<td>1</td>
<td>266</td>
</tr>
<tr>
<td>12</td>
<td>1534</td>
<td>1222</td>
<td>1383</td>
<td>12/0</td>
<td>1</td>
<td>274</td>
</tr>
</tbody>
</table>

Device, Part, Speed: XC7K70T,FBG484,-1 (ADVANCED 1.08a 2012-11-12)

### Table 2-6: Virtex-7

<table>
<thead>
<tr>
<th>Data Width</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAM 36 / 18</th>
<th>DSP48E1</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1112</td>
<td>972</td>
<td>1019</td>
<td>2/1</td>
<td>1</td>
<td>288</td>
</tr>
<tr>
<td>10</td>
<td>1234</td>
<td>1143</td>
<td>1201</td>
<td>5/0</td>
<td>1</td>
<td>281</td>
</tr>
<tr>
<td>12</td>
<td>1488</td>
<td>1256</td>
<td>1383</td>
<td>12/0</td>
<td>1</td>
<td>288</td>
</tr>
</tbody>
</table>

Device, Part, Speed: XC7V585T,FFG1157,-1 (PRODUCTION 1.08b 2012-11-12)

### Table 2-7: Virtex-6

<table>
<thead>
<tr>
<th>Data Width</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAM 36 / 18</th>
<th>DSP48E1</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1129</td>
<td>972</td>
<td>1019</td>
<td>2/1</td>
<td>1</td>
<td>281</td>
</tr>
<tr>
<td>10</td>
<td>1264</td>
<td>1095</td>
<td>1201</td>
<td>5/0</td>
<td>1</td>
<td>281</td>
</tr>
<tr>
<td>12</td>
<td>1443</td>
<td>1273</td>
<td>1383</td>
<td>12/0</td>
<td>1</td>
<td>258</td>
</tr>
</tbody>
</table>

Device, Part, Speed: XC6VLX75,FFG484,-1 (PRODUCTION 1.17 2012-11-12)

### Table 2-8: Spartan-6

<table>
<thead>
<tr>
<th>Data Width</th>
<th>LUT-FF Pairs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAM 16 / 8</th>
<th>DSP48A1</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1102</td>
<td>992</td>
<td>1071</td>
<td>5/0</td>
<td>2</td>
<td>110</td>
</tr>
<tr>
<td>10</td>
<td>1320</td>
<td>1098</td>
<td>1263</td>
<td>9/1</td>
<td>2</td>
<td>110</td>
</tr>
<tr>
<td>12</td>
<td>1529</td>
<td>1260</td>
<td>1455</td>
<td>24/0</td>
<td>2</td>
<td>118</td>
</tr>
</tbody>
</table>

Device, Part, Speed: XC6SLX25,FGG484,-2 (PRODUCTION 1.23 2012-11-12)
via the GUI with an AXI4-Lite control interface. The INTC_IF interface is present only when the core is configured via the GUI with the INTC interface enabled.

Table 2-9 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-9: 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>
</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. Refer to 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 (de-asserted) halts the operation of the core despite rising edges on the ACLK pin. Internal states are maintained, and output signal levels are held until ACLKEN is asserted again. When ACLKEN is de-asserted, core inputs are not sampled, except ARESETn, which supersedes ACLKEN. 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 de-asserted. 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 Edge 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 AXI Reference Guide (UG761) [Ref 1].

**AXI4-Stream Signal Names and Descriptions**

Table 2-10 describes the AXI4-Stream signal names and descriptions.
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 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 Image Edge Enhancement output m_axis_video_tdata is packed and padded to multiples of 8 bits as necessary, as seen in Figure 2-2. Zero padding the most significant bits is only necessary for 10 and 12 bit wide data.

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

### 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 2-3. 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 Edge Enhancement core driving s_axis_video_tready) may change value until the transaction completes (s_axis_video_tready and s_axis_video_tvalid ACLKEN are high on the rising edge of ACLK). Once asserted, s_axis_video_tvalid may only be
de-asserted after a transaction has completed. Transactions may not be retracted or aborted. In any cycle following a transaction, \texttt{s_axis_video_tvalid} can either be de-asserted or remain asserted to initiate a new transfer.

![Diagram](image)

\textbf{Figure 2-3: Example of READY/VALID Handshake, Start of a New Frame}

**Guidelines on Driving \texttt{m_axis_video_tready}**

The \texttt{m_axis_video_tready} signal may be asserted before, during or after the cycle in which the Image Edge 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.

\textit{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 2-3. SOF serves as a frame synchronization signal, which allows downstream cores to re-initialize, and detect the first pixel of a frame. The SOF signal may be asserted an arbitrary number of ACLK cycles before the first pixel value is presented on DATA, as long as a VALID is not asserted.

**End of Line Signals - \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 2-4.
Core Interfaces and Register Space

Control Interface

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

Constant Configuration

The constant configuration caters to users who will interface the core to a particular image sensor with a known, stationary resolution and use constant enhancement filter gains. 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 filter gains are hard coded into the core via the Image Edge 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 Edge Enhancement core can be controlled via the AXI4-Lite interface using read and write transactions to the Image Edge Enhancement register space.

Table 2-11: 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>
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_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 (de-asserted) 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 de-asserted, 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.

---

**Table 2-11: 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_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 de-asserted. 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 Edge Enhancement core uses only one timing related register, **ACTIVE_SIZE** (0x0020), which allows specifying the input frame dimensions. Also, the core has four core-specific register, **GAIN_H** (0x0100), **GAIN_V** (0x0104), **GAIN_D** (0x0108), and **GAIN_LAP** (0x010C) which allows specifying the gain of the enhancement filters.

**Table 2-12: Register Names and Descriptions**

<table>
<thead>
<tr>
<th>Address (hex)</th>
<th>Register Name</th>
<th>Access Type</th>
<th>Double Buffered</th>
<th>Default Value</th>
<th>Register Description</th>
</tr>
</thead>
</table>
| BASEADDR      | Control       | R/W         | No              | Power-on-Reset 0x0 | Bit 0: SW_ENABLE
Bit 1: REG_UPDATE
Bit 4: BYPASS
Bit 5: TEST_PATTERN
Bit 30: FRAME_SYNC_RESET
Bit 31: SW_RESET |
| 0x0000        | 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             | 0x0600A000    | 7-0: REVISION_NUMBER
11-8: PATCH_ID
15-12: VERSION_REVISION
23-16: VERSION_MINOR
31-24: VERSION_MAJOR |
| 0x0014        | SYSDEBUG0     | R           | N/A             | 0             | 0-31: Frame Throughput monitor |
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 Edge Enhancement core `ACTIVE_SIZE` and `GAIN` registers are double buffered. One set of registers (the processor registers) is directly accessed by the processor interface, while the other set (the active set) is actively used by the core. New values written to the processor registers will get copied over to the active set at the end of the AXI4-Stream frame, if and only if `REG_UPDATE` is set. Setting `REG_UPDATE` to 0 before updating multiple register values, then setting `REG_UPDATE` to 1 when updates are completed ensures all registers are updated simultaneously at the frame boundary without causing image tearing.

Bit 4 of the CONTROL register, `BYPASS`, switches the core to bypass mode if debug features are enabled. In bypass mode the Image Edge Enhancement core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output. Refer to 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 CONTROL register, TEST_PATTERN, switches the core to test-pattern generator mode if debug features are enabled. Refer to 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 CONTROL register, FRAME_SYNC_RESET and SW_RESET facilitate software reset. Setting SW_RESET reinitializes the core to GUI default values, all internal registers and outputs are cleared and held at initial values until SW_RESET is set to 0. The SW_RESET flag is not synchronized with the AXI4-Stream interfaces. Resetting the core while frame processing is in progress will cause image tearing. For applications where the software reset functionality is desirable, but image tearing has to be avoided a frame synchronized software reset (FRAME_SYNC_RESET) is available. Setting FRAME_SYNC_RESET to 1 will reset the core at the end of the frame being processed, or immediately if the core is between frames when the FRAME_SYNC_RESET was asserted. After reset, the FRAME_SYNC_RESET bit is automatically cleared, so the core can get ready to process the next frame of video as soon as possible. The default value of both RESET bits is 0. Core instances with no AXI4-Lite control interface can only be reset via the ARESETn pin.

**STATUS (0x0004) Register**

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

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

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

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

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

**ERROR (0x0008) Register**

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

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

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

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

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

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

**IRQ_ENABLE (0x000C) Register**

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

**Version (0x0010) Register**

Bit fields of the Version Register facilitate software identification of the exact version of the hardware peripheral incorporated into a system. The core driver can take advantage of this Read-Only value to verify that the software is matched to the correct version of the hardware. See Table 2-12 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. Refer to 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. Refer to 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. Refer to 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.

GAIN_H (0x0100) Register

The GAIN_H register contains the gain applied to the Horizontal Sobel filter. Allowed values are from 0 to 2 in increments of 1/16 represented by six unsigned bits with two integer bits and four fractional bits.

GAIN_V (0x0104) Register

The GAIN_V register contains the gain applied to the Vertical Sobel filter. Allowed values are from 0 to 2 in increments of 1/16 represented by six unsigned bits with two integer bits and four fractional bits.

GAIN_D (0x0108) Register

The GAIN_D register contains the gain applied to the left and right Diagonal Sobel filters. Allowed values are from 0 to 2 in increments of 1/16 represented by six unsigned bits with two integer bits and four fractional bits.
GAIN_LAP (0x010C) Register

The GAIN_LAP register contains the gain applied to the Laplacian filter. Allowed values are from 0 to 2 in increments of 1/16 represented by six unsigned bits with two integer bits and four fractional bits.

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 Edge 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 2-13.

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

Table 2-13: 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>
<tr>
<td>7</td>
<td>SOF Early</td>
</tr>
<tr>
<td>8</td>
<td>SOF Late</td>
</tr>
</tbody>
</table>

In a system integration tool, such as EDK, the interrupt controller INTC IP can be used to register the selected INTC_IF signals as edge triggered interrupt sources. The INTC IP
provides functionality to mask (enable or disable), as well as identify individual interrupt sources from software. Alternatively, for an external processor or MCU the user can custom build a priority interrupt controller to aggregate interrupt requests and identify interrupt sources.
Chapter 3

Designing with the Core

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

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

\[
\text{Horizontal Sobel} = \begin{bmatrix} 1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}
\]

Equation 3-1

\[
\text{Vertical Sobel} = \begin{bmatrix} 1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}
\]

Equation 3-2

\[
\text{Diagonal Sobels} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}
\]

Equation 3-3

The Laplacian is defined in Equation 3-4.

\[
\text{Laplacian} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}
\]

Equation 3-4
Defining Gains

The amount and direction of the edge enhancement can be controlled through programmable gains \( g_H \) (horizontal), \( g_V \) (vertical), \( g_D \) (diagonal), and \( g_{Lap} \) (Laplacian). Here, a vertical edge is defined as a feature running from top to bottom of an image. Similarly, a horizontal edge runs from left to right across the image. The diagonal direction covers both upper left to lower right and upper right to lower left diagonals.

Gains can be set to values in the range of 0.0 to 2.0. If a particular direction is not desired, that gain can be set to zero to eliminate emphasis in that direction. For example, if vertical edges do not need to be enhanced, the gain \( g_V \) should be set to zero.

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

If the total gain used \([(g_H + g_V + g_D + g_{Lap}) \times K]\) exceeds 1.0, clipping and clamping circuitry limits the enhancement of the edge content. Setting the gains with values greater than 1.0 allows over-enhancing the image to produce special effects like embossing.

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

---

General Design Guidelines

The Image Edge 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 Edge 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 Edge Enhancement core is part of a larger system such as the an Image Sensor Pipeline (ISP) System, as shown in Figure 3-1.
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 3-2.

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 Edge 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 will complete even if the video processing is stalled with ARESETn, ACLKEN or with the video clock not running.

**ACLKEN**

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

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

The **ACLKEN** pin facilitates:

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

**IMPORTANT:** To prevent transaction errors 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, 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 via AXI4-Stream interfaces, where only the master or the slave interface has an **ACLKEN** port that is not permanently tied high, the two interfaces should be connected via the AXI4-Stream Interconnect or AXI-FIFO cores to avoid data corruption (Figure 2-3).

**S_AXI_ACLKEN**

The **S_AXI_ACLKEN** is the clock enable signal for the AXI4-Lite interface only. Driving this signal low will only affect the AXI4-Lite interface and will not halt the video processing in the ACLK clock domain.

**ARESETn**

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

**CAUTION!** **ARESETn** is not synchronized internally to AXI4-Stream frame processing. Therefore, de-asserting **ARESETn** while a frame is being processed leads to image tearing.

The external reset pulse needs to be held for 32 ACLK cycles to reset the core. The **ARESETn** signal will only reset 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 resetting a system with multiple-clocks and corresponding reset signals, the reset generator must ensure that all reset signals are asserted/de-asserted long enough for all interfaces and clock-domains in all IP cores to correctly reinitialize.

**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 will reset the entire core including the AXI4-Lite and AXI4-Stream interfaces.

---

**System Considerations**

The Image Edge 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 will gather the image sensor timing signals. The AXI4-Lite control interface on the Video Timing Controller allows the system processor to read out the measured frame dimensions, and program all downstream cores, such as the Image Edge Enhancement, with the appropriate image dimensions.

If the target system uses only one, stationary image size, you may choose to consolidate the active-size and enhancement filter gains values, and create a constant configuration by removing the AXI4-Lite interface. This option allows reducing the core Slice footprint.

**Clock Domain Interaction**

The ARESETn and ACLKEN input signals will 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 will respond with an error if the core registers cannot be read or written within 128 S_AXI_ACLK clock cycles. The core registers cannot be read or written if the ARESETn signal is held low, if the ACLKEN signal is held low or if the ACLK signal is not connected or not running. If core register read does not complete, the AXI4-Lite read transaction will respond with 10 on the S_AXI_RRESP bus. Similarly, if a core register write does not complete, the AXI4-Lite write transaction will respond with 10 on the S_AXI_BRESP bus. The 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. 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 Edge 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.
C Model Reference

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

Features

- Bit-accurate with the Image Edge Enhancement v6.00a 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 Edge 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 Edge 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.
The latest version of the model is available for download on the Image Edge Enhancement product page at: 

---

**Unpacking and Model Contents**

Unzip the v_enhance_v6_00_a_bitacc_model.zip file, containing the bit accurate models for the Image Edge Enhancement IP Core. This creates the directory structure and files in Table 4-1.

**Table 4-1: Bit Accurate C Model Directory Structure and Files**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>README.txt</td>
<td>Release notes</td>
</tr>
<tr>
<td>doc/pg003_v_enhance.pdf</td>
<td>LogiCORE IP Image Edge Enhancement Product Guide</td>
</tr>
<tr>
<td>v_enhance_v6_00_a_bitacc_cmodel.h</td>
<td>Model header file</td>
</tr>
<tr>
<td>rgb_utils.h</td>
<td>Header file declaring the RGB image/video container type and support functions</td>
</tr>
<tr>
<td>yuv_utils.h</td>
<td>Header file declaring the YUV (.yuv) image file I/O functions</td>
</tr>
<tr>
<td>video_utils.h</td>
<td>Header file declaring the generalized image/video container type, I/O and support functions</td>
</tr>
<tr>
<td>run_bitacc_cmodel.c</td>
<td>Example code calling the C model</td>
</tr>
<tr>
<td>parsers.c</td>
<td>Code for reading configuration file</td>
</tr>
<tr>
<td>/examples</td>
<td>Example input files used by C model</td>
</tr>
<tr>
<td>enhance.cfg</td>
<td>Sample configuration file containing the core parameter settings</td>
</tr>
<tr>
<td>input_image.yuv</td>
<td>Sample test image</td>
</tr>
<tr>
<td>input_image.hdr</td>
<td>Sample test image header file</td>
</tr>
<tr>
<td>/lin64</td>
<td>Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms</td>
</tr>
<tr>
<td>liblp_v_enhance_v6_00_a_bitacc_cmodel.so</td>
<td>Model shared object library</td>
</tr>
<tr>
<td>libstlport.so.5.1</td>
<td>STL library, referenced by liblp_v_enhance_v6_00_a_bitacc_cmodel.so</td>
</tr>
<tr>
<td>/lin32</td>
<td>Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms</td>
</tr>
<tr>
<td>liblp_v_enhance_v6_00_a_bitacc_cmodel.so</td>
<td>Model shared object library</td>
</tr>
<tr>
<td>libstlport.so.5.1</td>
<td>STL library, referenced by liblp_v_enhance_v6_00_a_bitacc_cmodel.so</td>
</tr>
</tbody>
</table>
### Installation

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

- `libIp_v_enhance_v6_00_a_bitacc_cmodel.so`
- `libstlport.so.5.1`

### Software Requirements

The Image Edge Enhancement v6.00a C models were compiled and tested with the software listed in Table 4-2.

#### Table 4-2: 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_v6_00_a_bitacc_cmodel.h` file.
Before using the model, the structures holding the inputs, generics and output of the Image Edge Enhancement instance must be defined:

```c
struct xilinx_ip_v_enhance_v6_00_a_generics enhance_generics;
struct xilinx_ip_v_enhance_v6_00_a_inputs   enhance_inputs;
struct xilinx_ip_v_enhance_v6_00_a_outputs  enhance_outputs;
```

The declaration of these structures is in the `v_enhance_v6_00_a_bitacc_cmodel.h` file.

Table 4-3 lists the generic parameters taken by the Image Edge Enhancement v6.00a IP core bit accurate model, as well as the default values. For an actual instance of the core, these parameters can only be set in generation time through the CORE Generator™ GUI.

**Table 4-3: 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>DATA_WIDTH</td>
<td>int</td>
<td>8</td>
<td>8,10,12</td>
<td>Data width</td>
</tr>
</tbody>
</table>

Calling `xilinx_ip_v_enhance_v6_00_a_get_default_generics(&enhance_generics)` initializes the generics structure with the Image Edge Enhancement GUI defaults, listed in Table 4-3.

Direction gain 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 4-4).

**Table 4-4: 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.¹</td>
</tr>
<tr>
<td>gain_h</td>
<td>int</td>
<td>1</td>
<td>Allowed values are 0 to 2 in increments of 1/16</td>
<td>Horizontal gain</td>
</tr>
<tr>
<td>gain_v</td>
<td>int</td>
<td>1</td>
<td>Allowed values are 0 to 2 in increments of 1/16</td>
<td>Vertical gain</td>
</tr>
<tr>
<td>gain_d</td>
<td>int</td>
<td>1</td>
<td>Allowed values are 0 to 2 in increments of 1/16</td>
<td>Diagonal gain</td>
</tr>
<tr>
<td>gain_lap</td>
<td>int</td>
<td>1</td>
<td>Allowed values are 0 to 2 in increments of 1/16</td>
<td>Laplacian filter gain</td>
</tr>
</tbody>
</table>

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

The structure `enhance_inputs` defines the values of run time parameters and the actual input image. Calling `xilinx_ip_v_enhance_v6_00_a_get_default_inputs(&enhance_generics, &enhance_inputs)` initializes the input structure with the default values (see Table 4-4).
IMPORTANT: The video_in variable is not initialized because the initialization depends on the actual test image to be simulated. Chapter 4, C Model Example Code describes the initialization of the video_in structure.

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

```c
int xilinx_ip_v_enhance_v6_00_a_bitacc_simulate(
    struct xilinx_ip_v_enhance_v6_00_a_generics* generics,
    struct xilinx_ip_v_enhance_v6_00_a_inputs*   inputs,
    struct xilinx_ip_v_enhance_v6_00_a_outputs*  outputs).
```

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

```c
void xilinx_ip_v_enhance_v6_00_a_destroy(
    struct xilinx_ip_v_enhance_v6_00_a_inputs *input,
    struct xilinx_ip_v_enhance_v6_00_a_outputs *output).
```

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

## Image Edge Enhancement Input and Output Video Structure

Input images or video streams can be provided to the Image Edge Enhancement v6.00a reference model using the video_struct structure, defined in video_utils.h:

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

<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. Pertaining to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through all frames of the video stream. However different planes, such as y, u and v can have different dimensions.</td>
</tr>
<tr>
<td>cols</td>
<td>Number of columns per frame. Pertaining to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through all frames of the video stream. However different planes, such as y, u and v can have different dimensions.</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>
</tbody>
</table>
Using the C Model

The Image Edge Enhancement core supports the mode FORMAT_C444.

### Initializing the Image Edge Enhancement Input Video Structure

The easiest way to assign stimuli values to the input video structure is to initialize it with an image or video. The `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 \texttt{yuv8\_video\_struct} and general \texttt{video\_struct} type frames/videos is facilitated by these functions:

\begin{verbatim}
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 );
\end{verbatim}

\textbf{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.

\section*{Binary Image/Video Files}

The \texttt{video\_utils\_h} header file declares functions that help load and save generalized video files in raw, uncompressed format.

\begin{verbatim}
int read_video( FILE* infile, struct video\_struct* in\_video);
int write_video(FILE* outfile, struct video\_struct* out\_video);
\end{verbatim}

These functions serialize the \texttt{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.

\section*{Working with Video\_struct Containers}

The \texttt{video\_utils\_h} header file defines functions to simplify access to video data in \texttt{video\_struct}.

\begin{verbatim}
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);
\end{verbatim}

The \texttt{video\_planes\_per\_mode} function returns the number of component planes defined by the mode variable, as described in Table 4-6. The \texttt{video\_rows\_per\_plane} and \texttt{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 \texttt{in\_video} variable:

\begin{verbatim}
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++) {
        
      }
    }
  }
}
\end{verbatim}
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 Edge Enhancement C Model with Example Wrapper

Linux (32-bit and 64-bit)

To compile the example code, perform these steps:

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

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

   `libstlport.so.5.1`
libIp_v Enhance_v6 00_a_bitacc_cmodel.so

3. In the root directory, compile using the GNU C Compiler with this command:
   gcc -m32 -x c++ ../run_bitacc_cmodel.c ../gen_stim.c ../parsers.c -o run_bitacc_cmodel -L. -lIp_v Enhance_v6 00_a_bitacc_cmodel -Wl,-rpath,.
   gcc –m64 -x c++ ../run_bitacc_cmodel.c ../gen_stim.c ../parsers.c -o run_bitacc_cmodel -L. -lIp_v Enhance_v6 00_a_bitacc_cmodel -Wl,-rpath,.

Windows (32-bit and 64-bit)

The precompiled library v Enhance_v6 00_a_bitacc_cmodel.lib, and top-level demonstration code run_bitacc_cmodel.c should be compiled with an ANSI C compliant compiler under Windows. An example procedure is provided here using Microsoft Visual Studio.

1. In Visual Studio, create a new, empty Win32 Console Application project.

2. As existing items, add:
   a. libIp_v Enhance_v6 00_a_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_v6 00_a_bitacc_cmodel.h to the Header Files folder of the project

3. After the project is created and populated, it must be compiled and linked (built) to create a win32 executable. To perform the build step, select "Build Solution" from the Build menu. An executable matching the project name has been created either in the Debug or Release subdirectories under the project location based on whether "Debug" or "Release" has been selected in the "Configuration Manager" under the Build menu.
SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core
Constraining the Core
Detailed Example Design
Chapter 5

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.

Graphical User Interface

The Image Edge Enhancement core is easily configured to the user's specific needs through the Vivado design tools Graphical User Interface (GUI). This section provides a quick reference to the parameters that can be configured at generation time. Figure 5-1 shows the main Image Edge Enhancement screen.

Figure 5-1: Image Edge Enhancement Main Screen
The GUI displays a representation of the IP symbol on the left side, and the parameter assignments on the right side, which are described as follows:

- **Component Name**: The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_". The name `v_enhance_v6_00_a` cannot be used as a component name.

- **Video Component Width**: Specifies the bit width of input samples. Permitted values are 8, 10 and 12 bits.

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

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

- **INTC Interface**: When selected, the core will generate the optional `INTC_IF` port, which gives parallel access to signals indicating frame processing status and error conditions. For more information, refer to The Interrupt Subsystem in Chapter 2.

- **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 CORE Generator GUI as the default value for the lower half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the horizontal size of the frames the generated core instance is to process.
  - **Number of Scanlines per Frame**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the CORE Generator GUI as the default value for the upper half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the vertical size (number of lines) of the frames the generated core instance is to process.
  - **Maximum Number of 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 Active Pixels per Scan line** field as for constant mode the maximum number of pixels equals the active number of pixels.

- **Horizontal Sobel, Vertical Sobel, Diagonal Sobel, and Laplacian Gains:** Specifies the default gain to be applied for each filter. The possible values are 0.0 to 2.0 in increments of 1/16.
Constraining the Core

Required Constraints

The ACLK pin should be constrained at the desired pixel clock rate for your video stream. The S_AXI_ACLK pin should be constrained at the frequency of the AXI4-Lite subsystem. In addition to clock frequency, the following constraints should be applied to cover all clock domain crossing data paths.

XDC

```
set_max_delay -to [get_cells -hierarchical -match_style ucf "*U_VIDEO_CTRL*/*SYNC2PROCCLK_I*/data_sync_reg[0]"] -datapath_only 2
set_max_delay -to [get_cells -hierarchical -match_style ucf "*U_VIDEO_CTRL*/*SYNC2VIDCLK_I*/data_sync_reg[0]"] -datapath_only 2
```

Device, Package, and Speed Grade Selections

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

Clock Frequencies

The pixel clock (ACLK) frequency is the required frequency for this core. See Chapter 2, Maximum Frequencies. The S_AXI_ACLK maximum frequency is the same as the ACLK maximum.
Clock Management

The core automatically handles clock domain crossing between the ACLK (video pixel clock and AXI4-Stream) and the S_AXI_ACLK (AXI4-Lite) clock domains. The S_AXI_ACLK clock can be slower or faster than the ACLK clock signal, but must not be more than 128x faster than ACLK.

Clock Placement

There are no specific Clock placement requirements for this core.

Banking

There are no specific Banking rules for this core.

Transceiver Placement

There are no Transceiver Placement requirements for this core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for this core.
Detailed Example Design

No example design is available at the time for the LogiCORE IP Image Edge Enhancement v6.00a core.

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 tools. You are encouraged to make simple modifications to the configurations and observe the changes in the waveform.

Generating the Test Bench

After customizing the IP, right-click on the core instance in Sources pane and select Generate Output Products (Figure 7-1).
A pop-up window prompts you to select items to generate.

Click on Test Bench and make sure Action: Generate is selected.

The demonstration test bench package will be generated in the following directory (Figure 7-2):

<PROJ_DIR>/<PROJ_NAME>.srcs/sources_1/ip/<IP_INSTANCE_NAME>/demo_tb/
Directory and File Contents
The following files are expected to be generated in the demo 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 tb_<IP_instance_name>.

It instantiates the following modules:

- DUT
  The <IP> core instance under test.
- axi4lite_mst
The AXI4-Lite master module, which initiates AXI4-Lite transactions to program core registers.

- **axi4s_video_mst**

  The AXI4-Stream master module, which 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, refer to 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's 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, refer to C Model Reference.

- **ce_gen**

  Programmable Clock Enable (ACLKEN) generator.
Running the Simulation

There are two ways to run the demonstration test bench, after successfully generating the core output:

Option 1: Launch Simulation from the Vivado GUI

This runs the test bench with the AXI4-Stream Master producing ramp data as stimuli, and AXI4-Stream Slave set to passive mode.

1. Click Simulation Settings in the Flow Navigation window, change Simulation top module name to `tb_<IP_instance_name>`.
2. Click Run Simulation. XSIM launches and you should be able to see the signals.
3. You can also choose Modelsim for simulation by going to Project Settings and selecting Modelsim as the Target Simulator (Figure 7-3).

Option 2: Manually Compile and Run from the Simulation Environment

1. Add the generated test bench files to a new simulation set, along with the customized IP. For information on the location of generated test bench files, refer to Generating the Test Bench.
2. Setup the environment variables for Xilinx libraries
3. Compile the generated IP
4. Compile the test bench files
5. Run the simulation

**RECOMMENDED:** Change the default simulation time from 1000 ns to all to be able observe a full frame transaction.
SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core
Constraining the Core
Detailed Example Design
This chapter includes information about using Xilinx tools to customize and generate the core in the ISE® Design Suite environment.

**Graphical User Interface**

The Image Edge Enhancement core is easily configured to the user’s specific needs through the CORE Generator™ or EDK GUIs. This section provides a quick reference to the parameters that can be configured at generation time. Figure 8-1 shows the main Image Edge 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_v6_00_a` cannot be used as a component name.
- **Video Component Width**: Specifies the bit width of input samples. Permitted values are 8, 10, and 12 bits.

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

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

- **INTC Interface**: When selected, the core will generate the optional INTC_IF port, which gives parallel access to signals indicating frame processing status and error conditions. For more information, refer to The Interrupt Subsystem in Chapter 2.

- **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 CORE Generator GUI as the default value for the lower half-word of the ACTIVE_SIZE register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the horizontal size of the frames the generated core instance is to process.
  - **Number of Scanlines per Frame**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the CORE Generator GUI as the default value for the upper half-word of the ACTIVE_SIZE register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the vertical size (number of lines) of the frames the generated core instance is to process.
  - **Maximum Number of 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 Active Pixels per Scan line field as for constant mode the maximum number of pixels equals the active number of pixels.
  - **Horizontal Sobel, Vertical Sobel, Diagonal Sobel, and Laplacian Gains**: Specifies the default gain to be applied for each filter. The possible values are 0.0 to 2.0 in increments of 1/16.
Definitions of the EDK GUI controls are identical to the corresponding CORE Generator GUI functions.

---

**Parameter Values in the XCO File**

The following table defines valid entries for the XCO parameters. Parameters are not case sensitive. Xilinx strongly suggests that XCO parameters are not manually edited in the XCO.
file; instead, use the CORE Generator tool GUI to configure the core and perform range and parameter value checking.

**Table 8-1: XCO Parameters**

<table>
<thead>
<tr>
<th>XCO Parameter</th>
<th>Default</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>component_name</td>
<td>edge_enhancement</td>
<td>ASCII text using characters: a..z, 0..9 and &quot;_&quot; starting with a letter.</td>
</tr>
<tr>
<td>s_axis_video_data_width</td>
<td>8</td>
<td>8, 10, 12</td>
</tr>
<tr>
<td>has_axi4_lite</td>
<td>true</td>
<td>true, false</td>
</tr>
<tr>
<td>has_intc_if</td>
<td>false</td>
<td>true, false</td>
</tr>
<tr>
<td>has_debug</td>
<td>false</td>
<td>true, false</td>
</tr>
<tr>
<td>active_cols</td>
<td>1920</td>
<td>32 - 7680</td>
</tr>
<tr>
<td>active_rows</td>
<td>1080</td>
<td>32 - 7680</td>
</tr>
<tr>
<td>max_cols</td>
<td>1920</td>
<td>32 - 7680</td>
</tr>
<tr>
<td>gain_h</td>
<td>1</td>
<td>0 to 2 in 1/16th increments</td>
</tr>
<tr>
<td>gain_v</td>
<td>1</td>
<td>0 to 2 in 1/16th increments</td>
</tr>
<tr>
<td>gain_d</td>
<td>1</td>
<td>0 to 2 in 1/16th increments</td>
</tr>
<tr>
<td>gain_lap</td>
<td>1</td>
<td>0 to 2 in 1/16th increments</td>
</tr>
</tbody>
</table>

**Output Generation**

CORE Generator will output the core as a netlist that can be inserted into a processor interface wrapper or instantiated directly in an HDL design. The output is placed in the `<project director>`.

**File Details**

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

**Table 8-2: CORE Generator Output**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;component_name&gt;.xco</code></td>
<td>CORE Generator input file containing the parameters used to generate a core.</td>
</tr>
<tr>
<td><code>&lt;component_name&gt;.ngc</code></td>
<td>Binary Xilinx implementation netlist files containing the information required to implement the module in a Xilinx (R) FPGA.</td>
</tr>
<tr>
<td><code>&lt;component_name&gt;.vho</code></td>
<td>Template files containing code that can be used as a model for instantiating</td>
</tr>
<tr>
<td><code>&lt;component_name&gt;.veo</code></td>
<td>Template files containing code that can be used as a model for instantiating</td>
</tr>
</tbody>
</table>
### Table 8-2: CORE Generator Output (Cont'd)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;component_name&gt;.vhd</code></td>
<td>Structural simulation model</td>
</tr>
<tr>
<td><code>&lt;component_name&gt;.v</code></td>
<td></td>
</tr>
<tr>
<td><code>/doc/pg002_v_enhance.pdf</code></td>
<td>Core documents</td>
</tr>
<tr>
<td><code>/doc/v_enhance_v6_00_a_vinfo.html</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;component_name&gt;.asy</code></td>
<td>Graphical symbol information file. Used by the ISE tools and some third</td>
</tr>
<tr>
<td></td>
<td>party tools to create a symbol representing the core.</td>
</tr>
<tr>
<td><code>&lt;component_name&gt;_xmfd.tcl</code></td>
<td>ISE Project Navigator interface file. ISE uses this file to determine how</td>
</tr>
<tr>
<td></td>
<td>the files output by CORE Generator for the core can be integrated into your</td>
</tr>
<tr>
<td></td>
<td>ISE project.</td>
</tr>
<tr>
<td><code>&lt;component_name&gt;.gise</code></td>
<td>ISE Project Navigator support files. These are generated files and should</td>
</tr>
<tr>
<td><code>&lt;component_name&gt;.xise</code></td>
<td>not be edited directly.</td>
</tr>
<tr>
<td><code>&lt;component_name&gt;_readme.txt</code></td>
<td>Readme file for the IP.</td>
</tr>
<tr>
<td><code>&lt;component_name&gt;_flist.txt</code></td>
<td>Text file listing all of the output files produced when a customized core</td>
</tr>
<tr>
<td></td>
<td>was generated in the CORE Generator.</td>
</tr>
</tbody>
</table>
Chapter 9

Constraining the Core

Required Constraints

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

The \texttt{S\_AXI\_ACLK} pin should be constrained at the frequency of the AXI4-Lite subsystem.

In addition to clock frequency, the following constraints should be applied to cover all clock domain crossing data paths.

\textbf{UCF}

\begin{verbatim}
INST "+U\_VIDEO\_CTRL*/SYNC2PROCCLK\_I*/data_sync_reg[0]" TNM =
"async_clock_conv_FFDEST";
TIMESPEC "TS_async_clock_conv" = FROM FFS TO "async_clock_conv_FFDEST" 2 NS
DATAPATHONLY;
INST "+U\_VIDEO\_CTRLk*/SYNC2VIDCLK\_I*/data_sync_reg[0]" TNM =
"vid_async_clock_conv_FFDEST";
TIMESPEC "TS_vid_async_clock_conv" = FROM FFS TO "vid_async_clock_conv_FFDEST" 2 NS
DATAPATHONLY;
\end{verbatim}

Device, Package, and Speed Grade Selections

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

Clock Frequencies

The pixel clock (\texttt{ACLK}) frequency is the required frequency for this core. See Chapter 2, Maximum Frequencies. The \texttt{S\_AXI\_ACLK} maximum frequency is the same as the \texttt{ACLK} maximum.
Clock Management

The core automatically handles clock domain crossing between the ACLK (video pixel clock and AXI4-Stream) and the S_AXI_ACLK (AXI4-Lite) clock domains. The S_AXI_ACLK clock can be slower or faster than the ACLK clock signal, but must not be more than 128x faster than ACLK.

Clock Placement

There are no specific Clock placement requirements for this core.

Banking

There are no specific Banking rules for this core.

Transceiver Placement

There are no Transceiver Placement requirements for this core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for this core.
Chapter 10

Detailed Example Design

No example design is available at the time for the LogiCORE IP Image Edge Enhancement v6.00a core.

Demonstration Test Bench

A demonstration test bench is provided which enables core users to observe core behavior in a typical use scenario.

RECOMMENDED: Make simple modifications to the test conditions and observe the changes in the waveform.

Test Bench Structure

The top-level entity, \texttt{tb\_main.v}, instantiates the following modules:

- **DUT**
  
  The Image Edge Enhancement v6.00a core instance under test.

- **\texttt{axi4lite\_mst}**
  
  The AXI4-Lite master module, which initiates AXI4-Lite transactions to program core registers.

- **\texttt{axi4s\_video\_mst}**
  
  The AXI4-Stream master module, which opens the stimuli txt file and initiates AXI4-Stream transactions to provide stimuli data for the core.

- **\texttt{axi4s\_video\_slv}**
  
  The AXI4-Stream slave module, which opens the result txt file and verifies AXI4-Stream transactions from the core.
• ce_gen

Programmable Clock Enable (ACLKEN) generator

Running the Simulation

• Simulation using ModelSim for Linux:
  From the console, Type "source run_mti.sh".

• Simulation using iSim for Linux:
  From the console, Type "source run_isim.sh".

• Simulation using ModelSim for Windows:
  Double-click on "run_mti.bat" file.

• Simulation using iSim:
  Double-click on "run_isim.bat" file.

Directory and File Contents

The directory structure underneath the top-level folder is:

• **Expected**
  Contains the pre-generated expected/golden data used by the test bench to compare actual output data.

• **Stimuli**
  Contains the pre-generated input data used by the test bench to stimulate the core (including register programming values).

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

• **Src**
  Contains the .vhd simulation files and the .xco CORE Generator parameterization file of the core instance. The .vhd file is a netlist generated using CORE Generator. The .xco file can be used to regenerate a new netlist using CORE Generator.

The available core C-model can be used to generate stimuli and expected results for any user bmp image. For more information, refer to Chapter 4, C Model Reference.

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

• **isim_wave.wcfg**
  Waveform configuration for ISIM
• mti_wave.do:
  Waveform configuration for ModelSim

• run_isim.bat:
  Runscript for iSim in Windows

• run_isim.sh:
  Runscript for iSim in Linux

• run_mti.bat:
  Runscript for ModelSim in Windows

• run_mti.sh:
  Runscript for ModelSim in Linux
SECTION IV: APPENDICES

Verification, Compliance, and Interoperability
Migrating
Debugging
Additional Resources
Appendix A

Verification, Compliance, and Interoperability

Simulation

A highly parameterizable test bench was used to test the Image Edge 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 Edge 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 Edge 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. The core master (output) interface can work directly with any Xilinx Video core which consumes YCbCr 4:4:4 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 UG761 AXI Reference Guide.
Appendix B

Migrating

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

For a complete list of Vivado User and Methodology Guides, see the [Vivado Design Suite - 2012.4 User Guides web page](https://www.xilinx.com).

From version v5.01.a to v6.00a of the Image Edge Enhancement core the following significant changes took place:

- The Gain values are now in increments of 1/16th instead of 1/4. Two extra fractional bits need to be added to the Gain register values. For example, for a Gain of 0.25, the signal should now be 00.0100 instead of the previous value of 00.01 and a gain of 1 is 01.0000 instead of 01.00.

- The Gain registers are floating point values in the range of [0,2). These are represented as integer values by multiplying by 16 resulting in an integer value in the range of [0,32]. For example, a gain of 1 is now represented by the register value of 16 instead of the former value of 4.
Appendix C

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools. In addition, this appendix provides a step-by-step debugging process and a flow diagram to guide you through debugging the Image Edge Enhancement core.

The following topics are included in this appendix:

- Finding Help on Xilinx.com
- Debug Tools
- Hardware Debug
- Interface Debug
- AXI4-Stream Interfaces

Finding Help on Xilinx.com

To help in the design and debug process when using the Image Edge Enhancement, the Xilinx Support web page (www.xilinx.com/support) 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 Edge 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 (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.
Release Notes

Known issues for all cores, including the Image Edge Enhancement are described in the IP Release Notes Guide (XTP025).

Known Issues

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.

Contacting Technical Support

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

To contact Xilinx Technical Support:

2. Open a WebCase by selecting the WebCase link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- A block diagram of the video system that explains the video source, destination and IP (custom and Xilinx) used.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.
Appendix C:  Debugging

Debug Tools

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

Example Design

The Image Edge Enhancement is delivered with an example design that can be synthesized, complete with functional test benches. Information about the example design can be found in Chapter 6, Example Design for the Vivado™ Design Suite.

Core Wizard

The Image Edge Enhancement core is equipped with optional debugging features which aim to accelerate system bring-up, optimize memory and data-path architecture, and reduce time to market. The optional debug features can be turned on and off using the Include Debug Features checkbox on the GUI when an AXI4-Lite interface is present. Turning off debug features reduces the core footprint. Refer to the individual sections Core Bypass Option, Built in Test-Pattern Generator, and Throughput Monitors for a description of these debug features.

ChipScope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. The ChipScope Pro tool allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information for using the ChipScope Pro tool, see www.xilinx.com/tools/cspro.htm.

Vivado Lab Tools

Vivado Lab Tools inserts logic analyzer, bus 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.

Reference Boards

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

- 7 series evaluation boards
  - KC705
C-Model Reference

Please see *C Model Reference in Chapter 4* in this guide for tips and instructions for using the provided C-Model files to debug your design.

License Checkers

If the IP requires a license key, the key must be verified. The ISE and Vivado tool flows have a number of license check points for gating licensed IP through the flow. If the license check succeeds, the IP may continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- ISE flow: XST, NgdBui ld, Bitgen
- Vivado flow: Vivado Synthesis, Vivado Implementation, write_bitstream (Tcl command)

**IMPORTANT:** *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

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 ChipScope tool is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the ChipScope tool for debugging the specific problems.

Many of these common issues can also be applied to debugging design simulations. Details are provided on:

- General Checks
- Core Bypass Option
- Built-In Test Pattern Generator
- Throughput Monitors

**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 data paths.

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

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

Interface Debug

AXI4-Lite Interfaces

Table C-1 describes how to troubleshoot the AXI4-Lite interface.
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 <strong>ERROR</strong> register reads back set.</td>
<td>Bit 0 of the <strong>ERROR</strong> register, <strong>EOL_EARLY</strong>, 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 <strong>ACTIVE_SIZE</strong> register. If the value was provided by the Video Timing Controller core, read out <strong>ACTIVE_SIZE</strong> register value from the VTC core again, and make sure that the <strong>TIMING_LOCKED</strong> flag is set in the VTC core. Otherwise, using Chipscope, measure the number of active AXI4-Stream transactions between EOL pulses.</td>
</tr>
<tr>
<td>Bit 1 of the <strong>ERROR</strong> register reads back set.</td>
<td>Bit 1 of the <strong>ERROR</strong> register, <strong>EOL_LATE</strong>, indicates the number of pixels received between the last End-Of-Line (EOL) signal surpassed the value programmed into the <strong>ACTIVE_SIZE</strong> register. If the value was provided by the Video Timing Controller core, read out <strong>ACTIVE_SIZE</strong> register value from the VTC core again, and make sure that the <strong>TIMING_LOCKED</strong> flag is set in the VTC core. Otherwise, using Chipscope, measure the number of active AXI4-Stream transactions between EOL pulses.</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.

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

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<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 Chipscope, 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 CFA 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 CFA core cannot send data downstream, and the internal FIFOs are full.</td>
</tr>
<tr>
<td>m_axis_video_tvalid stuck low, the downstream core is not receiving data</td>
<td>• 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.</td>
</tr>
<tr>
<td>Generated SOF signal (m_axis_video_tuser0) signal misplaced.</td>
<td>Check the ERROR register.</td>
</tr>
<tr>
<td>Generated EOL signal (m_axis_video_tl_last) signal misplaced.</td>
<td>Check the ERROR register.</td>
</tr>
<tr>
<td>Data samples lost between Upstream core and the CFA core. Inconsistent EOL and/or SOF periods received.</td>
<td>• 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 the CFA core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the CFA ACLK pin lower than the reported Fmax reached?</td>
</tr>
<tr>
<td>Data samples lost between Downstream core and the CFA core. Inconsistent EOL and/or SOF periods received.</td>
<td>• 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 the CFA core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the CFA ACLK pin lower than the reported Fmax reached?</td>
</tr>
</tbody>
</table>

**Other Interfaces**

Table C-3 describes how to troubleshoot third-party interfaces.
### 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 <code>TDATA</code> signal is in accordance to <a href="#">Data Interface in Chapter 2</a>. If misaligned: In HDL, break up the <code>TDATA</code> vector to constituent components and manually connect the slave and master interface sides. In EDK, create a new vector for the slave side <code>TDATA</code> connection. In the MPD file, manually assign components of the master-side <code>TDATA</code> vector to sections of the new vector.</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 <code>TDATA</code> signal color component assignments described in <a href="#">Data Interface in Chapter 2</a> in mind, there are no guarantees that the software correctly identifies bits corresponding to color components. Verify that the color component logical addressing <code>TDATA</code> is in alignment with the data format expected by the software drivers reading/writing external memory. If misaligned: In HDL, break up the <code>TDATA</code> vector to constituent components, and manually connect the slave and master interface sides. In EDK, create a new vector for the slave side <code>TDATA</code> connection. In the MPD file, manually assign components of the master-side <code>TDATA</code> vector to sections of the new vector.</td>
</tr>
</tbody>
</table>
The software API is provided to allow easy access to the Image Edge Enhancement AXI4-Lite registers defined in Table 2-9. To utilize the API functions, the following two header files must be included in the user C code:

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

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

For examples on API function calls and integration into a user application, the drivers subdirectory of the pCore contains a file, `example.c`, in the `enhance_v6_00_a0_a/example` subfolder. This file is a sample C program that demonstrates how to use the Image Edge Enhancement pCore API.

### Table D-1: Image Edge Enhancement Driver Function Definitions

<table>
<thead>
<tr>
<th>Function Name and Parameterization</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENHANCE_Enable (uint32 BaseAddress)</td>
<td>Enables a Image Edge Enhancement instance.</td>
</tr>
<tr>
<td>ENHANCE_Disable (uint32 BaseAddress)</td>
<td>Disables a Image Edge Enhancement instance.</td>
</tr>
<tr>
<td>ENHANCE_Reset (uint32 BaseAddress)</td>
<td>Immediately resets a Image Edge Enhancement instance. The core stays in reset until the RESET flag is cleared.</td>
</tr>
<tr>
<td>ENHANCE_ClearReset (uint32 BaseAddress)</td>
<td>Clears the reset flag of the core, which allows it to re-sync with the input video stream and return to normal operation.</td>
</tr>
<tr>
<td>ENHANCE_FSync_Reset (uint32 BaseAddress)</td>
<td>Resets a Image Edge Enhancement instance at the end of the current frame being processed, or immediately if the core is not currently processing a frame.</td>
</tr>
<tr>
<td>ENHANCE_ReadReg (uint32 BaseAddress, uint32 RegOffset)</td>
<td>Returns the 32-bit unsigned integer value of the register. Read the register selected by RegOffset.</td>
</tr>
</tbody>
</table>
**Software Reset**

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

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

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

**Double Buffering**

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

If multiple register values are changed during frame processing, simple double buffering would not guarantee that all register updates would take effect at the beginning of the same frame. Using a semaphore mechanism, the RegUpdateEnable() and

<table>
<thead>
<tr>
<th>Function Name and Parameterization</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENHANCE_WriteReg (uint32 BaseAddress, uint32 RegOffset, uint32 Data)</td>
<td>Write the register selected by RegOffset. Data is the 32-bit value to write to the register.</td>
</tr>
<tr>
<td>ENHANCE_RegUpdateEnable (uint32 BaseAddress)</td>
<td>Enables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.</td>
</tr>
<tr>
<td>ENHANCE_RegUpdateDisable (uint32 BaseAddress)</td>
<td>Disables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.</td>
</tr>
</tbody>
</table>
RegUpdateDisable() functions allows synchronous commitment of register changes. The Image Edge Enhancement core will start using the updated ACTIVE_SIZE and GAIN values only if the REGUPDATE flag of the CONTROL register is set (1), after the next Start-Of-Frame signal (s_axis_video_tuser) is received. Therefore, it is recommended to disable the register update before writing multiple double-buffered registers, then enable register update when register writes are completed.

Reading and Writing Registers

Each software register that is defined in Table 2-12 has a constant that is defined in enhance.h which is set to the offset for that register listed in Table D-2.

**RECOMMENDED:** Use the predefined register names instead of register values in the application software when accessing core registers. This will prevent any future updates to the Image Edge Enhancement drivers which may change register locations from negatively affecting the application dependent on the Image Edge Enhancement driver.

<table>
<thead>
<tr>
<th>Constant Name Definition</th>
<th>Value</th>
<th>Target Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENHANCE_CONTROL</td>
<td>0x0000</td>
<td>CONTROL</td>
</tr>
<tr>
<td>ENHANCE_STATUS</td>
<td>0x0004</td>
<td>STATUS</td>
</tr>
<tr>
<td>ENHANCE_ERROR</td>
<td>0x0008</td>
<td>ERROR</td>
</tr>
<tr>
<td>ENHANCE_IRQ_ENABLE</td>
<td>0x000C</td>
<td>IRQ_ENABLE</td>
</tr>
<tr>
<td>ENHANCE_VERSION</td>
<td>0x0010</td>
<td>VERSION</td>
</tr>
<tr>
<td>ENHANCE_SYSDEBUG0</td>
<td>0x0014</td>
<td>SYSDEBUG0</td>
</tr>
<tr>
<td>ENHANCE_SYSDEBUG1</td>
<td>0x0018</td>
<td>SYSDEBUG1</td>
</tr>
<tr>
<td>ENHANCE_SYSDEBUG2</td>
<td>0x001C</td>
<td>SYSDEBUG2</td>
</tr>
<tr>
<td>ENHANCE_ACTIVE_SIZE</td>
<td>0x0020</td>
<td>ACTIVE_SIZE</td>
</tr>
<tr>
<td>ENHANCE_GAIN_H</td>
<td>0x0100</td>
<td>GAIN_H</td>
</tr>
<tr>
<td>ENHANCE_GAIN_V</td>
<td>0x0104</td>
<td>GAIN_V</td>
</tr>
<tr>
<td>ENHANCE_GAIN_D</td>
<td>0x0108</td>
<td>GAIN_D</td>
</tr>
<tr>
<td>ENHANCE_GAIN_LAP</td>
<td>0x010C</td>
<td>GAIN_LAP</td>
</tr>
</tbody>
</table>
Appendix E

Additional Resources

Xilinx Resources

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


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:


Solution Centers

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

References

These documents provide supplemental material useful with this user guide:

1. UG761 AXI Reference Guide
2. Vivado Design Suite Migration Methodology Guide (UG911)
3. Vivado™ Design Suite user documentation
Technical Support

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

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

- New Features
- Resolved Issues
- Known Issues

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>10/19/2011</td>
<td>1.0</td>
<td>Initial Xilinx release of Product Guide, replacing DS753 and UG831.</td>
</tr>
<tr>
<td>4/24/2012</td>
<td>2.0</td>
<td>Updated for core version. Added Zynq-7000 devices, added AXI4-Stream interfaces, deprecated GPP interface.</td>
</tr>
<tr>
<td>07/25/2012</td>
<td>3.0</td>
<td>Updated for core version. Added Vivado information.</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. Added Vivado test bench.</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. Updated resource numbers, Debugging appendix, and gain registers.</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; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.

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