Table of Contents

IP Facts

Chapter 1: Overview
Feature Summary ............................................................... 5
Applications ........................................................................ 6
Licensing and Ordering Information ......................................... 6

Chapter 2: Product Specification
Standards ............................................................................. 7
Performance ......................................................................... 7
Resource Utilization ............................................................... 8
Core Interfaces and Register Space ......................................... 9

Chapter 3: Designing with the Core
General Design Guidelines ...................................................... 23
Color-Space Conversion Background ........................................ 24
Clock, Enable, and Reset Considerations ................................. 30
System Considerations ........................................................... 32

Chapter 4: C Model Reference
Installation and Directory Structure ......................................... 35
Using the C-Model .................................................................... 37
Compiling with the YCrCb to RGB C-Model ............................... 43

Chapter 5: Customizing and Generating the Core
Vivado Integrated Design Environment (IDE) ............................ 45
Output Generation ................................................................. 49

Chapter 6: Constraining the Core
Required Constraints ............................................................. 50
Device, Package, and Speed Grade Selections ............................ 50
Clock Frequencies ................................................................... 50
Clock Management ................................................................ 50
Introduction

The Xilinx LogiCORE™ IP YCrCb to RGB Color-Space Converter core is a simplified 3x3 matrix multiplier converting three input color samples to three output samples in a single clock cycle. The optimized structure uses only four XtremeDSP™ slices by taking advantage of the dependencies between coefficients in the conversion matrix of most YCrCb 4:4:4 or YUV 4:4:4 to RGB standards.

Features

- Built-in support for:
  - SD (ITU 601)
  - HD (ITU 709) PAL
  - HD (ITU 709) NTSC
  - YUV
- Support for user-defined conversion matrices
- AXI4-Stream data interfaces
- Optional AXI4-Lite control interface
- Supports 8, 10, 12 and 16-bit 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.

LogiCORE IP Facts Table

<table>
<thead>
<tr>
<th>Core Specifics</th>
<th>Supported Device Family(1)</th>
<th>Zynq™-7000, Artix™-7, Virtex®-7, Kintex™-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported User Interfaces</td>
<td>AXI4-Lite, AXI4-Stream (2)</td>
<td></td>
</tr>
<tr>
<td>Resources</td>
<td>See Table 2-1 through Table 2-4</td>
<td></td>
</tr>
</tbody>
</table>

Provided with Core

| Documentation | Product Guide |
| Design Files | Encrypted RTL |
| Example Design | Not Provided |
| Test Bench | Verilog (3) |
| Constraints File | Not Provided |
| Simulation Models | Encrypted RTL, VHDL or Verilog Structural, C-Model (4) |
| Supported Software Drivers | Standalone |

Tested Design Flows (5)

| Design Entry Tools | Vivado™ Design Suite |
| Simulation | Mentor Graphics Questa SIM, Vivado Simulator |
| Synthesis Tools | Vivado Synthesis |

Support

Provided by Xilinx, Inc.

1. For a complete listing of supported devices, see the Vivado IP Catalog.
2. Video protocol as defined in the Video IP: AXI Feature Adoption section of (UG761) AXI Reference Guide [Ref 7].
4. Standalone driver details can be found in the SDK directory (<install_directory>/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from //wiki.xilinx.com.
5. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
Overview

A color space is a mathematical representation of a set of colors. The two most popular color models are:

- RGB or R’G’B’, gamma corrected RGB, used in computer graphics
- YIQ, YUV and YCrCb used in video systems

These color spaces are directly related to the intuitive notions of hue, saturation and brightness.

All color spaces can be derived from the RGB information supplied by devices such as cameras and scanners. Different color spaces have historically evolved for different applications. In each case, a color space was chosen for application-specific reasons.

The convergence of computers, the Internet and a wide variety of video devices, all using different color representations, is forcing the digital designer today to convert between them. The objective is to have all inputs converted to a common color space before algorithms and processes are executed. Converters are useful for a number of markets, including image and video processing.

Feature Summary

The YCrCb to RGB Color-Space Converter core transforms YCrCb 4:4:4 or YUV 4:4:4 video data into RGB video data. The core supports 4 common format conversions as well as custom mode that allows for a user-defined transform. The core is capable of a maximum resolution of 7680 columns by 7680 rows with 8, 10, 12, or 16 bits per pixel and supports the bandwidth necessary for High-definition (1080p60) resolutions in all Xilinx FPGA device families. Higher resolutions can be supported in Xilinx high-performance device families.

You can configure and instantiate the core from the Vivado Design Suite. Core functionality may be controlled dynamically with an optional AXI4-Lite interface.
Applications

• Post-processing core for image data
• Video surveillance
• Video conferencing
• Machine vision
• Other imaging applications

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no cost under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the YCrCb to RGB Color-Space Converter 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 YCrCb to RGB Color-Space Converter 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 7] for additional information.

Performance

The following sections detail the performance characteristics of the YCrCb to RGB Color-Space Converter core.

Maximum Frequencies

This section contains typical clock frequencies for the target devices. The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the device, using a different version of Xilinx tools and other factors. See Table 2-1 through Table 2-4 for device-specific information.

Latency

The processing latency of the core is shown in the following equation:

\[
\text{Latency} = 9 + 1(\text{if has clipping}) + 1(\text{if has clamping})
\]

This code evaluates to 11 clock cycles for typical cases (unless in "custom" mode the clipping and/or clamping circuits are not used).

Throughput

The YCrCb to RGB Color Space Converter core outputs one YCbCr 4:4:4 sample per clock cycle.
Resource Utilization

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

Table 2-1 through Table 2-4 were generated using Vivado Design Suite with the AXI4-Lite interface, INTC_IF, and the Debug Features disabled.

**Table 2-1: Kintex-7 FPGA and Zynq-7000 Devices with Kintex Based Programmable Logic**

<table>
<thead>
<tr>
<th>Data Width</th>
<th>Slice FFs</th>
<th>Slice LUTs</th>
<th>LUT6-FF pairs</th>
<th>DSPs</th>
<th>Clock Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>237</td>
<td>248</td>
<td>274</td>
<td>8</td>
<td>234</td>
</tr>
<tr>
<td>10</td>
<td>275</td>
<td>272</td>
<td>295</td>
<td>8</td>
<td>234</td>
</tr>
<tr>
<td>12</td>
<td>313</td>
<td>311</td>
<td>336</td>
<td>8</td>
<td>234</td>
</tr>
<tr>
<td>16</td>
<td>389</td>
<td>375</td>
<td>410</td>
<td>8</td>
<td>234</td>
</tr>
</tbody>
</table>

**Table 2-2: Artix-7 FPGA and Zynq-7000 Devices with Artix Based Programmable Logic**

<table>
<thead>
<tr>
<th>Data Width</th>
<th>Slice FFs</th>
<th>Slice LUTs</th>
<th>LUT6-FF pairs</th>
<th>DSPs</th>
<th>Clock Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>237</td>
<td>248</td>
<td>268</td>
<td>8</td>
<td>188</td>
</tr>
<tr>
<td>10</td>
<td>275</td>
<td>271</td>
<td>293</td>
<td>8</td>
<td>188</td>
</tr>
<tr>
<td>12</td>
<td>313</td>
<td>311</td>
<td>338</td>
<td>8</td>
<td>196</td>
</tr>
<tr>
<td>16</td>
<td>389</td>
<td>374</td>
<td>405</td>
<td>8</td>
<td>172</td>
</tr>
</tbody>
</table>

**Table 2-3: Virtex-7 FPGA Performance**

<table>
<thead>
<tr>
<th>Data Width</th>
<th>Slice FFs</th>
<th>Slice LUTs</th>
<th>LUT6-FF pairs</th>
<th>DSPs</th>
<th>Clock Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>237</td>
<td>247</td>
<td>268</td>
<td>8</td>
<td>234</td>
</tr>
<tr>
<td>10</td>
<td>275</td>
<td>272</td>
<td>294</td>
<td>8</td>
<td>234</td>
</tr>
<tr>
<td>12</td>
<td>313</td>
<td>311</td>
<td>343</td>
<td>8</td>
<td>234</td>
</tr>
<tr>
<td>16</td>
<td>389</td>
<td>375</td>
<td>402</td>
<td>8</td>
<td>234</td>
</tr>
</tbody>
</table>
Core Interfaces and Register Space

Port Descriptions

The YCrCb to RGB Color-Space Converter 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 YCrCb2RGB core. Some signals are optional and not present for all configurations of the core. The AXI4-Lite interface and the IRQ pin are present only when the core is configured via the GUI with an AXI4-Lite control interface. The INT_IF interface is present only when the core is configured via the GUI with the INTC interface enabled.

<table>
<thead>
<tr>
<th>Data Width</th>
<th>Slice FFs</th>
<th>Slice LUTs</th>
<th>LUT6-FF pairs</th>
<th>DSPs</th>
<th>Clock Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>237</td>
<td>248</td>
<td>272</td>
<td>8</td>
<td>226</td>
</tr>
<tr>
<td>10</td>
<td>275</td>
<td>272</td>
<td>295</td>
<td>8</td>
<td>226</td>
</tr>
<tr>
<td>12</td>
<td>313</td>
<td>311</td>
<td>338</td>
<td>8</td>
<td>234</td>
</tr>
<tr>
<td>16</td>
<td>389</td>
<td>375</td>
<td>405</td>
<td>8</td>
<td>234</td>
</tr>
</tbody>
</table>
Table 2-5 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-5: Common Interface Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACLK</td>
<td>In</td>
<td>1</td>
<td>Video Core Clock</td>
</tr>
<tr>
<td>ACLKEN</td>
<td>In</td>
<td>1</td>
<td>Video Core Active High Clock Enable</td>
</tr>
<tr>
<td>ARESETn</td>
<td>In</td>
<td>1</td>
<td>Video Core Active Low Synchronous Reset</td>
</tr>
<tr>
<td>INTC_IF</td>
<td>Out</td>
<td>6</td>
<td>Optional External Interrupt Controller Interface. Available only when INTC_IF is selected on GUI.</td>
</tr>
<tr>
<td>IRQ</td>
<td>Out</td>
<td>1</td>
<td>Optional Interrupt Request Pin. Available only when AXI4-Lite interface is selected on GUI.</td>
</tr>
</tbody>
</table>
The ACLK, ACLKEN and ARESETn signals are shared between the core and the AXI4-Stream data interfaces. The AXI4-Lite control interface has its own set of clock, clock enable and reset pins: s_AXI_ACLK, s_AXI_ACLKEN and s_AXI_ARESETn. Refer to The Interrupt Subsystem for a 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 YCrCb2RGB 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 (UG761) AXI Reference Guide [Ref 7].

**AXI4-Stream Signal Names and Descriptions**

Table 2-6 describes the AXI4-Stream signal names and descriptions.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_axis_video_tdata</td>
<td>In</td>
<td>24,32,40,48</td>
<td>Input Video Data</td>
</tr>
<tr>
<td>s_axis_video_tvalid</td>
<td>In</td>
<td>1</td>
<td>Input Video Valid Signal</td>
</tr>
<tr>
<td>s_axis_video_tready</td>
<td>Out</td>
<td>1</td>
<td>Input Ready</td>
</tr>
</tbody>
</table>
**Table 2-6: AXI4-Stream Data Interface Signal Descriptions**

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

**Video Data**

The AXI4-Stream interface specification restricts TDATA widths to integer multiples of 8 bits. Therefore, 10 and 12 bit sensor data must be padded with zeros on the MSB to form Nx8 bit wide vector before connecting to s_axis_video_tdata. Padding does not affect the size of the core.

For example, YCC data on the YCrCb2RGB input s_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 only necessary for 10 and 12 bits wide data.

![Figure 2-2: YCrCb Data Encoding on s_axis_video_tdata](X12683)

Similarly, RGB data on the YCrCb2RGB output m_axis_video_tdata is packed and padded to multiples of 8 bits as necessary, as seen in Figure 2-3. Zero padding the most significant bits is only necessary for 10 and 12 bit wide data.

![Figure 2-3: RGB Data Encoding on m_axis_video_tdata](X12683)

**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 2-4. 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 YCrCb2RGB core driving `s_axis_video_tready`) may change value until the transaction completes (`s_axis_video_tready, s_axis_video_tvalid` ACLKEN high on the rising edge of ACLK). Once asserted, `s_axis_video_tvalid` may only be de-asserted after a transaction has completed. Transactions may not be retracted or aborted. In any cycle following a transaction, `s_axis_video_tvalid` can either be de-asserted or remain asserted to initiate a new transfer.

![Diagram](image)

*Figure 2-4: Example of READY/VALID Handshake, Start of a New Frame*

**Guidelines on Driving m_axis_video_tready**

The `m_axis_video_tready` signal may be asserted before, during or after the cycle in which the YCrCb2RGB core asserted `m_axis_video_tvalid`. The assertion of `m_axis_video_tready` may be dependent on the value of `m_axis_video_tvalid`. A slave that can immediately accept data qualified by `m_axis_video_tvalid`, should pre-assert its `m_axis_video_tready` signal until data is received. Alternatively, `m_axis_video_tready` can be registered and driven the cycle following `VALID` assertion. It is recommended that the AXI4-Stream slave should drive `READY` independently, or pre-assert `READY` to minimize latency.

**Start of Frame Signals - m_axis_video_tuser0, s_axis_video_tuser0**

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 2-4. 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 - \textit{m\_axis\_video\_tlast, s\_axis\_video\_tlast}

The End-Of-Line signal, physically transmitted over the AXI4-Stream TLAST signal, marks the last pixel of a line. The EOL pulse is 1 valid transaction wide, and must coincide with the last pixel of a scan-line, as seen in Figure 2-5.

![Control Interface Diagram](image)

\textbf{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.

\textbf{Constant Configuration}

The constant configuration caters to users who will use the core in one setup that will not need to change over time. In constant configuration the image resolution (number of active pixels per scan line and the number of active scan lines per frame), and the other core parameters are hard coded into the core via the YCrCb2RGB core GUI. Since there is no AXI4-Lite interface, the core is not programmable, but can be reset, enabled, or disabled using the \texttt{ARESETn} and \texttt{ACLKEN} ports.

\textbf{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 YCrCb2RGB core can be controlled via the AXI4-Lite interface using read and write transactions to the YCrCb2RGB register space.
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.

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 \texttt{S\_AXI\_ACLKEN} is de-asserted, AXI4-Lite interface inputs are not sampled, except \texttt{S\_AXI\_ARESETn}, which supersedes \texttt{S\_AXI\_ACLKEN}. The AXI4-Stream interfaces signals are not affected by the \texttt{S\_AXI\_ACLKEN}.

\textbf{S\_AXI\_ARESETn}

The \texttt{S\_AXI\_ARESETn} pin is an active-low, synchronous reset input for the AXI4-Lite interface. \texttt{S\_AXI\_ARESETn} supersedes \texttt{S\_AXI\_ACLKEN}, and when set to 0, the core resets at the next rising edge of \texttt{S\_AXI\_ACLK} even if \texttt{S\_AXI\_ACLKEN} is de-asserted. The \texttt{S\_AXI\_ARESETn} signal must be synchronous to the \texttt{S\_AXI\_ACLK} and must be held low for a minimum of 32 clock cycles of the slowest clock. The \texttt{S\_AXI\_ARESETn} input is resynchronized to the \texttt{ACLK} clock domain. The AXI4-Stream interfaces and core signals are also reset by \texttt{S\_AXI\_ARESETn}.

\textbf{Register Space}

The standardized Xilinx Video IP register space is partitioned to control-, timing-, and core specific registers. The YCrCb2RGB core uses only one timing related register, \texttt{ACTIVE\_SIZE} (0x0020), which allows specifying the input frame dimensions. The core has nine core specific registers that control Matrix coefficients, the data offsets and the clip and clamp values.

\textit{Table 2-8: 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>
<tbody>
<tr>
<td>0x0000</td>
<td>CONTROL</td>
<td>R/W</td>
<td>N</td>
<td>No AXI4-Lite IF: 0x1 Power-on-Reset: 0x0</td>
<td>Bit 0: SW_ENABLE Bit 1: REG_UPDATE Bit 4: BYPASS(^{(1)}) Bit 5: TEST_PATTERN(^{(1)}) Bit 30: FRAME_SYNC_RESET (1: reset) Bit 31: SW_RESET (1: reset)</td>
</tr>
<tr>
<td>0x0004</td>
<td>STATUS</td>
<td>R/W</td>
<td>No</td>
<td>0</td>
<td>Bit 0: PROC_STARTED Bit 1: EOF Bit 16: SLAVE_ERROR</td>
</tr>
<tr>
<td>0x0008</td>
<td>ERROR</td>
<td>R/W</td>
<td>No</td>
<td>0</td>
<td>Bit 0: SLAVE_EOL_EARLY Bit 1: SLAVE_EOL_LATE Bit 2: SLAVE_SOF_EARLY Bit 3: SLAVE_SOF_LATE</td>
</tr>
<tr>
<td>0x000C</td>
<td>IRQ_ENABLE</td>
<td>R/W</td>
<td>No</td>
<td>0</td>
<td>16-0: Interrupt enable bits corresponding to STATUS bits</td>
</tr>
</tbody>
</table>
Table 2-8: Register Names and Descriptions

<table>
<thead>
<tr>
<th>Address (hex) BASEADDR +</th>
<th>Register Name</th>
<th>Access Type</th>
<th>Double Buffered</th>
<th>Default Value</th>
<th>Register Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0010</td>
<td>VERSION</td>
<td>R</td>
<td>N/A</td>
<td>0x070000000</td>
<td>7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR</td>
</tr>
<tr>
<td>0x0014</td>
<td>SYSDEBUG0</td>
<td>R</td>
<td>N/A</td>
<td>0</td>
<td>0-31: Frame Throughput monitor(1)</td>
</tr>
<tr>
<td>0x0018</td>
<td>SYSDEBUG1</td>
<td>R</td>
<td>N/A</td>
<td>0</td>
<td>0-31: Line Throughput monitor(2)</td>
</tr>
<tr>
<td>0x001C</td>
<td>SYSDEBUG2</td>
<td>R</td>
<td>N/A</td>
<td>0</td>
<td>0-31: Pixel Throughput monitor(4)</td>
</tr>
<tr>
<td>0x0020</td>
<td>ACTIVE_SIZE</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified via GUI</td>
<td>12-0: Number of Active Pixels per Scanline 28-16: Number of Active Lines per Frame</td>
</tr>
<tr>
<td>0x0100</td>
<td>RGBMAX</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified via GUI</td>
<td>15:0: RGB clipping value</td>
</tr>
<tr>
<td>0x0104</td>
<td>RGBMIN</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified via GUI</td>
<td>15:0: RGB clamping value</td>
</tr>
<tr>
<td>0x0108</td>
<td>ROFFSET</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified via GUI</td>
<td>31:0: Red offset compensation</td>
</tr>
<tr>
<td>0x010C</td>
<td>GOFFSET</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified via GUI</td>
<td>31:0: Green offset compensation</td>
</tr>
<tr>
<td>0x0110</td>
<td>BOFFSET</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified via GUI</td>
<td>31:0: Blue offset compensation</td>
</tr>
<tr>
<td>0x0114</td>
<td>ACOEF</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified via GUI</td>
<td>17:0: ACOEF, BCOEF, CCOEF, DCOEF are derived from CA, CB, CC and CD, by calculating the inverse multiplication matrix and representing as a 17-bit fixed point number.</td>
</tr>
<tr>
<td>0x0118</td>
<td>BCOEF</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified via GUI</td>
<td></td>
</tr>
<tr>
<td>0x011C</td>
<td>CCOEF</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified via GUI</td>
<td></td>
</tr>
<tr>
<td>0x0120</td>
<td>DCOEF</td>
<td>R/W</td>
<td>Yes</td>
<td>Specified via GUI</td>
<td></td>
</tr>
</tbody>
</table>

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

**CONTROL (0x00000) 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. The default value of SW enable is 1 (enabled) for the Constant configuration. 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 YCrCb2RGB core ACTIVE_SIZE and core specific 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 YCrCb2RGB 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 progress will cause image tearing. For applications where the soft-ware reset functionality is desirable, but image tearing has to be avoided a frame synchronized software reset (FRAME_SYNC_RESET) is available. Setting FRAME_SYNC_RESET to 1 will reset the core at the end of the frame being processed, or immediately if the core is between frames when the FRAME_SYNC_RESET was asserted. After reset, the FRAME_SYNC_RESET bit is automatically cleared, so the core can get ready to process the next frame of video as soon as possible. The default value of both RESET bits is 0. Core instances with no AXI4-Lite control interface can only be reset via the ARESETn pin.

**STATUS (0x0004) Register**

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

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

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 4 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 inverted 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 2-8 for more information.
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 Debug Tools in Appendix C for more information.

SYSDEBUG1 (0x0018) Register

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

SYSDEBUG2 (0x001C) Register

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

ACTIVE_SIZE (0x0020) Register

The ACTIVE_SIZE register encodes the number of active pixels per scan line and the number of active scan lines per frame. The lower half-word (bits 12:0) encodes the number of active pixels per scan line. Supported values are between 32 and the value provided in the Maximum number of pixels per scan line field in the GUI. The upper half-word (bits 28:16) encodes the number of active pixels per scan line. 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.

RGBMAX (0x0100) Register

The RGBMAX register holds the maximum value allowed on the Red, Green and Blue channels of the output. If the output data is greater than this value, then this value replaces it on the output. This register is only valid if "Outputs Clipped" is selected in the core parameterization GUI.

RGBMIN (0x0104) Register

The YMin register holds the minimum value allowed on the Red, Green and Blue channels of the output. If the output data is less than this value, then this value replaces it on the output. This register is only valid if "Outputs Clamped" is selected in the core parameterization GUI.
ROFFSET (0x0108) Register
The ROFFSET register holds the offset compensation value for the Red channel.

GOFFSET(0x010C) Register
The GOFFSET register holds the offset compensation value for the Green channel.

BOFFSET (0x0110) Register
The BOFFSET register holds the offset compensation value for the Blue channel.

ACOEF (0x0114) Register
The ACOEF register holds the transformed CA coefficient expressed as an 18.16 floating point number.

BCOEF (0x0118) Register
The BCOEF register holds the transformed CB coefficient expressed as an 18.16 floating point number.

CCOEF (0x011C) Register
The CCOEF register holds the transformed CC coefficient expressed as an 18.16 floating point number.

DCOEF (0x0120) Register
The DCOEF register holds the transformed CD coefficient expressed as an 18.16 floating point number.

The Interrupt Subsystem

STATUS register bits can trigger interrupts so embedded application developers can quickly identify faulty inter-faces or incorrectly parameterized cores in a video system. Irrespective of whether the AXI4-Lite control interface is present or not, the YCrCb2RGB 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 Ena**ble 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 2-9.

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

**Table 2-9: 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>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>Video over AXI4-Stream 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, 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

General Design Guidelines

The YCrCb2RGB core converts YCrCb 4:4:4 (or YUV 4:4:4) video data into RGB video data. The core processes samples provided via an AXI4-Stream slave interface, outputs pixels via an AXI4-Stream master interface, and can be controlled via an optional AXI4-Lite interface. The YCrCb2RGB block cannot change the input/output image sizes, the input and output pixel clock rates, or the frame rate. It is recommended that the YCrCb2RGB core is used in conjunction with the Video In to AXI4-Stream and Video Timing Controller cores. The Video Timing Controller core measures the timing parameters, such as number of active scan lines, number of active pixels per scan line of the image sensor. The Video In to AXI4-Stream core converts a standard parallel clocked video interface with syncs and or blanks to AXI4-Stream Video protocol as defined in the Video IP: AXI Feature Adoption section of the (UG761) AXI Reference Guide [Ref 7].

Typically, the YCrCb to RGB core is part of an Image Sensor Pipeline (ISP) System, as shown in Figure 3-1.
Color-Space Conversion Background

RGB Color Space

The red, green and blue (RGB) color space is widely used throughout computer graphics. Red, green and blue are three primary additive colors: individual components are added together to form a desired color, and are represented by a three dimensional, Cartesian coordinate system, as shown in Figure 3-2.

Table 3-1 presents the RGB values for 100% saturated color bars, a common video test signal.

<table>
<thead>
<tr>
<th>Normal Range</th>
<th>White</th>
<th>Yellow</th>
<th>Cyan</th>
<th>Green</th>
<th>Magenta</th>
<th>Red</th>
<th>Blue</th>
<th>Black</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0 to 255</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0 to 255</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0 to 255</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>255</td>
</tr>
</tbody>
</table>

The RGB color space is the most prevalent choice for computer graphics because color displays use red, green and blue to create the desired color. Also, a system that is designed using the RGB color space can take advantage of a large number of existing software algorithms.

However, RGB is not very efficient when dealing with real-world images. All three components need equal bandwidth to generate arbitrary colors within the RGB color cube. Also, processing an image in the RGB color space is usually not the most efficient method. For example, to modify the intensity or color of a given pixel, all three RGB values must be read, modified and written back to the frame buffer. If the system had access to the image stored in the intensity and color format, the process would be faster.

R'G'B' Color Space

While the RGB color space is ideal to represent computer graphics, 8-bit linear-light coding performs poorly for images to be viewed [Ref 2]. It is necessary to have 12 or 14 bits per component to achieve excellent quality. The best perceptual use is made of a limited number of bits by using nonlinear coding that mimics the nonlinear response of human vision. In video, JPEG, MPEG, computing, digital photography, and many other domains, a nonlinear transfer function is applied to the RGB signals to give nonlinearly coded gamma-corrected components, denoted with symbols R'G'B'. Excellent image quality can be obtained with 10-bit nonlinear coding with a transfer function similar to that of Rec. 709 [Ref 4] or RGB.
YUV Color Space

The YUV color space is used by the analog PAL, NTSC and SECAM color video/TV standards. In the past, black and white systems used only the luminance (Y) information. Chrominance information (U and V) was added in such a way that a black and white receiver can still display a normal black and white picture.

YCrCb (or YCbCr) Color Space

The YCrCb or YCbCr color space was developed as part of the ITU-R BT.601 [Ref 3] during the development of a world-wide digital component video standard. YCbCr is a scaled, offset version of the YUV color space. Y has a nominal range of 16-235; Cb and Cr have a nominal range of 16-240. There are several YCbCr sampling formats, such as 4:4:4, 4:2:2 and 4:2:0.

Conversion Equations

Derivation of Conversion Equations

To generate the luminance (Y, or gray value) component, biometric experiments were employed to measure how the human eye perceives the intensities of the red, green and blue colors. Based on these experiments, optimal values for coefficients CA and CB were determined, such that:

\[ Y = CA \times R + (1 - CA - CB) \times G + CB \times B \]  

Equation 3-1

Actual values for CA and CB differ slightly in different standards.

Conversion from the RGB color space to luminance and chrominance (differential color components) could be described with the following equation:
Coefficients CA and CB are chosen between 0 and 1, which guarantees that the range of Y is constrained between the maximum and the minimum RGB values permitted, RGB\textsubscript{max} and RGB\textsubscript{min} respectively.

In most practical implementations, the range of the luminance and chrominance components should be equal. There are two ways to accomplish this: the chrominance components (B-Y and R-Y) can be normalized (compressed and offset compensated), or values above and below the luminance range can be clipped/clamped.

Both clipping and dynamic range compression results in loss of information; however, the introduced artifacts are different. To leverage differences in the input (RGB) range, different standards choose different tradeoffs between clipping and normalization.

The YCrCb to RGB Color-Space Converter core supports only the conversions that fit the following general form:

\[
\begin{bmatrix}
  Y \\
  R - Y \\
  B - Y \\
\end{bmatrix} = \begin{bmatrix}
  CA & 1 - CA - CB & CB \\
  1 - CA & CA + CB - 1 & -CB \\
  -CA & CA + CB - 1 & 1 - CB \\
\end{bmatrix} \begin{bmatrix}
  R \\
  G \\
  B \\
\end{bmatrix}
\]

\textit{Equation 3-2}

\[
\begin{bmatrix}
  R \\
  G \\
  B \\
\end{bmatrix} = \begin{bmatrix}
  Y \\
  CR \\
  CB \\
\end{bmatrix} = \begin{bmatrix}
  CA & 1 - CA - CB & CB \\
  CC(1 - CA) & CC(CA + CB - 1) & CC(-CB) \\
  CD(-CA) & CD(CA + CB - 1) & CD(1 - CB) \\
\end{bmatrix} \begin{bmatrix}
  R \\
  G \\
  B \\
\end{bmatrix} + \begin{bmatrix}
  O_Y \\
  O_Cr \\
  O_Cb \\
\end{bmatrix} 
\]

\textit{Equation 3-3}

CC and CD allow dynamic range compression for B-Y and R-Y, and constants O\textsubscript{Y} and O\textsubscript{CR} facilitate offset compensation for the resulting CB and CR. To avoid arithmetic under- and overflows while converting from the RGB to the YCrCb domain, with RGB values in the [0.1] range, a choice for CC and CD is:

\[
CC = \frac{1}{2(1 - CA)} \quad CD = \frac{1}{2(1 - CB)}
\]

\textit{Equation 3-4}

The YCrCb to RGB core facilitates both range de-compression and optional clipping and clamping. Range, offset, clipping and clamping levels are parameterizable.

By inverting the transformation matrix in \textit{Equation 3-3}, the transformation from the YCrCb color space to the RGB color space can be defined as:

\[
\begin{bmatrix}
  R \\
  G \\
  B \\
\end{bmatrix} = \begin{bmatrix}
  1 & 1/CC & 0 \\
  -CA & -CB & 0 \\
  CC(1 - CA - CB) & CD(1 - CA - CB) & 1/CD \\
\end{bmatrix} \begin{bmatrix}
  Y - O_Y \\
  CR - O_Cr \\
  CB - O_Cb \\
\end{bmatrix}
\]

\textit{Equation 3-5}
Hardware Implementation

The YCrCb to RGB color-space transformation (Equation 3-5) can be expressed as:

\[ R = Y - O_Y + ACOEF(C_R - O_{Cr}) \]
\[ G = Y - O_Y + BCOEF(C_R - O_{Cr}) + CCOEF(C_B - O_{Cb}) \]
\[ B = Y - O_Y + DCOEF(C_B - O_{Cb}) \]

Equation 3-6
Equation 3-7
Equation 3-8

This cannot efficiently utilize the MADD capabilities of XtremeDSP slices. As offsets and coefficients are constants, the preceding equations can be rewritten as:

\[ R = ACOEF \cdot C_R + ROFFSET + Y \]
\[ G = BCOEF \cdot C_R + CCOEF \cdot C_B + GOFFSET + Y \]
\[ B = DCOEF \cdot C_B + BOFFSET + Y \]

Equation 3-9
Equation 3-10
Equation 3-11

This can be directly mapped to the architecture shown in Figure 3-3. The blue and gray boxes represent logic blocks, which are always implemented using XtremeDSP slices.

Assigning Values to Design Parameters

The following section specifies parameter values for some widely used standards. Most parameter values, except for COEF and OFFSET parameters, can be assigned from Table 3-2, Table 3-3 and Table 3-4 directly. These parameters have to be calculated, scaled and rounded before assigning integer values to corresponding VHDL parameters, using the following equations:
Coefficients are passed to the core in 17 bits wide two's complement format. Y, Cr and Cb are passed as DATA_WIDTH bits wide unsigned integers. MWIDTH is preset to Min(32, DATA_WIDTH+17).

\[
\text{ACOEF} = \frac{1}{CC} \quad \text{Equation 3-12}
\]

\[
\text{BCOEF} = \frac{-CA}{CC(1 - CA - CB)} \quad \text{Equation 3-13}
\]

\[
\text{CCOEF} = \frac{-CB}{CD(1 - CA - CB)} \quad \text{Equation 3-14}
\]

\[
\text{DCOEF} = \frac{1}{CD} \quad \text{Equation 3-15}
\]

\[
\text{ROUNDING_CONST} = 2^{\text{MWIDTH} - \text{DATA_WIDTH}} \quad \text{Equation 3-16}
\]

\[
\text{ROFFSET} = \text{ROUNDING_CONST} - (\text{ACOEF} \cdot \text{CROFFSET} + \text{YOFFSET}) \cdot \text{SCALE_M} \quad \text{Equation 3-17}
\]

\[
\text{GOFFSET} = \text{ROUNDING_CONST} - ((\text{BOEF} + \text{CROFFSET}) + (\text{CCOEF} \cdot \text{CBOFFSET}) + \text{YOFFSET}) \cdot \text{SCALE_N} \quad \text{Equation 3-18}
\]

\[
\text{BOFFSET} = \text{ROUNDING_CONST} - (\text{DCOEF} \cdot \text{CBOFFSET} + \text{YOFFSET}) \cdot \text{SCALE_M} \quad \text{Equation 3-19}
\]

\[
\text{SCALE_M} = 2^{\text{MWIDTH} - \text{IWIDTH} - 17} \quad \text{Equation 3-20}
\]

**ITU 601 (SD) and 709 - 1125/60 (NTSC) Standard Conversion Coefficients**

<table>
<thead>
<tr>
<th>Coefficient/ Parameter</th>
<th>Range</th>
<th>16-240</th>
<th>16-235</th>
<th>0-255</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td></td>
<td>0.299</td>
<td>0.2568</td>
<td></td>
</tr>
<tr>
<td>CB</td>
<td></td>
<td>0.114</td>
<td>0.0979</td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>0.713</td>
<td>0.7295</td>
<td>0.5910</td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>0.564</td>
<td></td>
<td>0.5772</td>
<td></td>
</tr>
<tr>
<td>YOFFSET</td>
<td>2^{\text{DATA_WIDTH}-4}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CB/CR</td>
<td>2^{\text{DATA_WIDTH}-1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YMAX</td>
<td>240*2^{\text{DATA_WIDTH}-8}</td>
<td>235*2^{\text{DATA_WIDTH}-8}</td>
<td>2^{\text{DATA_WIDTH}-1}</td>
<td></td>
</tr>
<tr>
<td>CMAX</td>
<td>240*2^{\text{DATA_WIDTH}-8}</td>
<td>235*2^{\text{DATA_WIDTH}-8}</td>
<td>2^{\text{DATA_WIDTH}-1}</td>
<td></td>
</tr>
<tr>
<td>YMIN</td>
<td>16*2^{\text{DATA_WIDTH}-8}</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CMIN</td>
<td>16*2^{\text{DATA_WIDTH}-8}</td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Standard ITU 709 (HD) 1250/50 (PAL)**
**Table 3-3: Parameterization Values for the PAL HD (ITU 709) Standard**

<table>
<thead>
<tr>
<th>Coefficient/Parameter</th>
<th>Input range</th>
<th>16-240</th>
<th>16-235</th>
<th>0-255</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td></td>
<td>0.2126</td>
<td></td>
<td>0.1819</td>
</tr>
<tr>
<td>CB</td>
<td></td>
<td>0.0722</td>
<td></td>
<td>0.0618</td>
</tr>
<tr>
<td>CC</td>
<td></td>
<td>0.6350</td>
<td>0.6495</td>
<td>0.6495</td>
</tr>
<tr>
<td>CD</td>
<td></td>
<td>0.5389</td>
<td></td>
<td>0.5512</td>
</tr>
<tr>
<td>YOFFSET</td>
<td></td>
<td></td>
<td></td>
<td>2 \text{DATA WIDTH-4}</td>
</tr>
<tr>
<td>OFFSET</td>
<td></td>
<td></td>
<td></td>
<td>2 \text{DATA WIDTH-1}</td>
</tr>
<tr>
<td>YMAX</td>
<td></td>
<td>240*2 \text{DATA WIDTH-8}</td>
<td>235*2 \text{DATA WIDTH-8}</td>
<td>2 \text{DATA WIDTH-1}</td>
</tr>
<tr>
<td>CMAX</td>
<td></td>
<td>240*2 \text{DATA WIDTH-8}</td>
<td>235*2 \text{DATA WIDTH-8}</td>
<td>2 \text{DATA WIDTH-1}</td>
</tr>
<tr>
<td>YMIN</td>
<td></td>
<td>16*2 \text{DATA WIDTH-8}</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CMIN</td>
<td></td>
<td>16*2 \text{DATA WIDTH-8}</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

**YUV Standard**

**Table 3-4: Parameterization Values for the YUV Standard**

<table>
<thead>
<tr>
<th>Coefficient/Parameter</th>
<th>Value</th>
<th>16-240</th>
<th>16-235</th>
<th>0-255</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>0.299</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CB</td>
<td>0.114</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>0.877283</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>0.492111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YOFFSET</td>
<td>2 \text{DATA WIDTH-4}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFFSET</td>
<td>2 \text{DATA WIDTH-1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YMAX</td>
<td>240*2 \text{DATA WIDTH-8}</td>
<td>235*2 \text{DATA WIDTH-8}</td>
<td>2 \text{DATA WIDTH-1}</td>
<td></td>
</tr>
<tr>
<td>CMAX</td>
<td>240*2 \text{DATA WIDTH-8}</td>
<td>235*2 \text{DATA WIDTH-8}</td>
<td>2 \text{DATA WIDTH-1}</td>
<td></td>
</tr>
<tr>
<td>YMIN</td>
<td>16*2 \text{DATA WIDTH-8}</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CMIN</td>
<td>16*2 \text{DATA WIDTH-8}</td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

**Clipping and Clamping**

**Output Quantization Noise**

Coefficients CC and CD in Equation 3-3 allow standard designers to trade off output quantization and clipping noise. Actual noise inserted depends on the probability statistics of the Cb and Cr variables, but in general if CC and CD are larger than the maximum values calculated in Equation 3-4, output values may clip, introducing clipping noise. However, the
lower CC and CD values are chosen, the worse Cb and Cr values will use the available dynamic range, thus introducing more quantization noise. Therefore, the designer’s task is to equalize output quantization and clipping noise insertion by carefully choosing CC and CD values based on knowing the statistics of Cb and Cr values. For instance, when probabilities of extreme chrominance values are small, it is beneficial to increase CC and CD values, as the extra noise inserted by occasional clipping is less than the gain in average signal power (and thus SQNR).

Output Clipping Noise

If coefficients CC and CD in Equation 3-3 are larger than the maximum values calculated in Equation 3-4, Cr and Cb output values may get larger (overflow) than the maximum or smaller (underflow) than minimum value the output representation can carry. If overflow occurs and the design does not have clipping logic, binary values wrap around and insert substantial noise to the output. If clamping/clipping logic is used, output values saturate and less noise is introduced, as shown in Figure 3-4. Use of clipping and clamping increases slice count of the design by approximately 6*DATA_WIDTH slices.

If a targeted standard limits output of values to a predefined range other than those of binary representation, such as ITU-R BT.601-5 [Ref 3], use of clipping and clamping logic facilitates constraining output values to the predefined range by setting RGB_max and RGB_min values according to the standard specifications.

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 3-5.
Clock, Enable, and Reset Considerations

**S_AXI_ACLK**

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

**ACLKEN**

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

ACLKEN may not be synchronized internally to AXI4-Stream frame processing therefore de-asserting ACLKEN for extended periods of time may lead to image tearing.

The ACLKEN pin facilitates:

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

**IMPORTANT:** When ACLKEN (clock enable) pins are used (toggled) in conjunction with a common clock source driving the master and slave sides of an AXI4-Stream interface, to prevent transaction errors the ACLKEN pins associated with the master and slave component interfaces must also be driven by the same signal (Figure 2-2).

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

**S_AXI_ACLKEN**

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

**ARESETn**

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

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

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

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

**S_AXI_ARESETn**

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

---

**System Considerations**

When using the YCrCb to RGB, it needs to be configured for the actual frame size and the proper color-space conversion to operate properly. To gather the frame size information from the video, it can be connected to the Video In to AXI-Stream input and the Video Timing Controller. The timing detector logic in the Video Timing Controller will gather the video 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 YCrCb to RGB, with the appropriate image dimensions.
If the target system uses only one configuration of the YCrCb2RGB core (i.e. does not need to be reprogrammed ever), you may choose to create a constant configuration by removing the AXI4-Lite interface. This reduces the core Slice footprint.

**Clock Domain Interaction**

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

**Programming Sequence**

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

**Error Propagation and Recovery**

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

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

Similarly, when EOL is detected early, the output EOL signal is generated early, terminating the previous line immediately. When EOL is detected late, the output EOL signal is
generated according to the programmed values. Extra pixels from the previous line are dropped until the input EOL is captured.
Chapter 4

C Model Reference

Installation and Directory Structure

This chapter contains information for installing the YCrCb to RGB Color-Space Converter C-Model, and describes the file contents and directory structure.


Software Requirements

The YCrCb to RGB Color-Space Converter v7.0 C-models were compiled and tested with the following software versions.

Table 4-1: Supported Systems and Software Requirements

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

Installation

The installation of the C-Model requires updates to the PATH variable, as described below.

Linux

Ensure that the directory in which the libIp_v_ycrcb2rgb_v7_0_bitacc_cmodel.so and libstlport.so.5.1 files are located is in your $LD_LIBRARY_PATH environment variable.
C-Model File Contents

Unzipping the v_ycrcb2rgb_v7_0_bitacc_model.zip file creates the following directory structures and files which are described in Table 4-2.

Table 4-2: C-Model Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/lin</td>
<td>Pre-compiled bit accurate ANSI C reference model for simulation on 32-bit Linux Platforms</td>
</tr>
<tr>
<td>libIp_ycrcb2rgb_v7_0_bitacc_cmodel.lib</td>
<td>YCrCb to RGB Color-Space Converter v7.0 model shared object library (Linux platforms only)</td>
</tr>
<tr>
<td>libstlport.so.5.1</td>
<td>STL library, referenced by the YCrCb to RGB Color-Space Converter library (Linux platforms only)</td>
</tr>
<tr>
<td>run_bitacc_cmodel</td>
<td>Pre-compiled bit accurate executable for simulation on 32-bit Linux Platforms</td>
</tr>
<tr>
<td>/lin64</td>
<td>Pre-compiled bit accurate ANSI C reference model for simulation on 64-bit Linux Platforms</td>
</tr>
<tr>
<td>libIp_ycrcb2rgb_v7_0_bitacc_cmodel.lib</td>
<td>YCrCb to RGB Color-Space Converter v7.0 model shared object library (Linux platforms only)</td>
</tr>
<tr>
<td>libstlport.so.5.1</td>
<td>STL library, referenced by the YCrCb to RGB Color-Space Converter library (Linux platforms only)</td>
</tr>
<tr>
<td>run_bitacc_cmodel</td>
<td>Pre-compiled bit accurate executable for simulation on 32-bit Linux Platforms</td>
</tr>
<tr>
<td>/nt</td>
<td>Pre-compiled bit accurate ANSI C reference model for simulation on 32-bit Windows Platforms</td>
</tr>
<tr>
<td>libIp_ycrcb2rgb_v7_0_bitacc_cmodel.lib</td>
<td>Pre-compiled library file for win32 compilation</td>
</tr>
<tr>
<td>run_bitacc_cmodel.exe</td>
<td>Pre-compiled bit accurate executable for simulation on 32-bit Windows Platforms</td>
</tr>
<tr>
<td>/nt64</td>
<td>Pre-compiled bit accurate ANSI C reference model for simulation on 64-bit Windows Platforms</td>
</tr>
<tr>
<td>libIp_ycrcb2rgb_v7_0_bitacc_cmodel.lib</td>
<td>Pre-compiled library file for win32 compilation</td>
</tr>
<tr>
<td>run_bitacc_cmodel.exe</td>
<td>Pre-compiled bit accurate executable for simulation on 64-bit Windows Platforms</td>
</tr>
<tr>
<td>README.txt</td>
<td>Release notes</td>
</tr>
<tr>
<td>pg014_v_ycrcb2rgb.pdf</td>
<td>YCrCb to RGB Color-Space Converter Core Product Guide</td>
</tr>
<tr>
<td>v_ycrcb2rgb_v7_0_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>bmp_utils.h</td>
<td>Header file declaring the bitmap (.bmp) image file I/O functions</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_ycrcb2rgb_v7_0_bitacc_cmodel.h` file. Before using the model, the structures holding the inputs, generics and output of the YCrCb to RGB Color-Space Converter instance must be defined:

```c
struct xilinx_ip_v_ycrcb2rgb_v7_0_generics generics;
struct xilinx_ip_v_ycrcb2rgb_v7_0_inputs inputs;
struct xilinx_ip_v_ycrcb2rgb_v7_0_outputs outputs;
```

The declaration of these structures is in the `v_ycrcb2rgb_v7_0_bitacc_cmodel.h` file. 

Table 4-3 lists the generic parameters taken by the YCrCb to RGB Color-Space Converter v4.0 IP core bit accurate model, as well as the 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>ACTIVE_COLS</td>
<td>int</td>
<td>1920</td>
<td>0-7680</td>
<td>Active # of Columns</td>
</tr>
<tr>
<td>ACTIVE_ROWS</td>
<td>int</td>
<td>1080</td>
<td>0-7680</td>
<td>Active # of Rows</td>
</tr>
<tr>
<td>COEF_IN</td>
<td>ycrcb_coef_in</td>
<td></td>
<td></td>
<td>YCC coefficient input structure</td>
</tr>
<tr>
<td>COEF_OUT</td>
<td>ycrcb_coef_out</td>
<td></td>
<td></td>
<td>YCC coefficient output structure</td>
</tr>
<tr>
<td>RGBMAX</td>
<td>int</td>
<td>240</td>
<td>0 - 2OWIDTH-1</td>
<td>Clipping value for the R, G and B Channels</td>
</tr>
<tr>
<td>RGBMIN</td>
<td>int</td>
<td>16</td>
<td>0 - 2OWIDTH-1</td>
<td>Clipping value for the R, G and B Channels</td>
</tr>
<tr>
<td>HAS_CLIP</td>
<td>int</td>
<td>1</td>
<td>0,1</td>
<td>Determines if Clipping is performed on the output data</td>
</tr>
<tr>
<td>HAS_CLAMP</td>
<td>int</td>
<td>1</td>
<td>0,1</td>
<td>Determines if Clamping is performed on the output data</td>
</tr>
</tbody>
</table>

Table 4-2: C-Model Files (Cont’d)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>Test_stimuli.bmp</td>
<td>32x32 sample test image</td>
</tr>
<tr>
<td>run_bittacc_cmodel.c</td>
<td>Example code calling the C-Model</td>
</tr>
</tbody>
</table>
Calling `xilinx_ip_v_ycrcb2rgb_v7_0_get_default_generics(&generics)` initializes the generics structure with the default value.

The `inputs` structure defines the actual input image. For the description of the input video structure, see `Input and Output Video Structures`.

Calling `xilinx_ip_v_ycrcb2rgb_v7_0_get_default_inputs(&generics, &inputs)` initializes the input video structure before it can be assigned an image or video sequence using the memory allocation or file I/O functions provided in the BMP, RGB or video utility functions.
**Note:** The `video_in` variable is not initialized to point to a valid image/video container, as the container size depends on the actual test image to be simulated. The initialization of the `video_in` structure is described in Initializing the Input Video Structure.

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

```c
int xilinx_ip_v_ycrcb2rgb_v7_0_bitacc_simulate(
    struct xilinx_ip_v_ycrcb2rgb_v7_0_generics* generics,
    struct xilinx_ip_v_ycrcb2rgb_v7_0_inputs*   inputs,
    struct xilinx_ip_v_ycrcb2rgb_v7_0_outputs*  outputs);
```

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

```c
void xilinx_ip_v_ycrcb2rgb_v7_0_destroy(
    struct xilinx_ip_v_ycrcb2rgb_v7_0_inputs *input,
    struct xilinx_ip_v_ycrcb2rgb_v7_0_outputs *output);
```

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

**Input and Output Video Structures**

Input images or video streams can be provided to the YCrCb to RGB Color-Space Converter v4.0 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 4-6: Member Variables of the Video Structure*

<table>
<thead>
<tr>
<th>Member Variable</th>
<th>Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>frames</td>
<td>Number of video/image frames in the data structure.</td>
</tr>
<tr>
<td>rows</td>
<td>Number of rows per frame. 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

**Table 4-6:** Member Variables of the Video Structure (Cont’d)

<table>
<thead>
<tr>
<th>mode</th>
<th>Contains information about the designation of data planes. Named constants to be assigned to mode are listed in Table 4-7.</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Set of five pointers to three dimensional arrays containing data for image planes. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col].</td>
</tr>
</tbody>
</table>

**Table 4-7:** Named Video Modes with Corresponding Planes and Representations

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

¹ The Color Space Conversion core C model supports FORMAT_C444 for input data and FORMAT_RGB for output data.

**Initializing the 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, bmp_util.h and video_util.h header files packaged with the bit accurate C models contain functions to facilitate file I/O.

**Bitmap Image Files**

The header bmp_utils.h declares functions that help access files in Windows Bitmap format (http://en.wikipedia.org/wiki/BMP_file_format). However, this format limits color depth to a maximum of 8-bits per pixel, and operates on images with three planes (R,G,B). Consequently, the following functions operate on arguments type rgb8_video_struct, which is defined in rgb_utils.h. Also, both functions support only true-color, non-indexed formats with 24-bits per pixel.

```c
int write_bmp(FILE *outfile, struct rgb8_video_struct *rgb8_video);
int read_bmp(FILE *infile, struct rgb8_video_struct *rgb8_video);
```
Exchanging data between `rgb8_video_struct` and general `video_struct` type frames/videos is facilitated by these functions:

```c
int copy_rgb8_to_video(struct rgb8_video_struct* rgb8_in,
                        struct video_struct* video_out);
int copy_video_to_rgb8(struct video_struct* video_in,
                        struct rgb8_video_struct* rgb8_out);
```

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

### Binary Image/Video Files

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

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

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

### YUV Image Files

The `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_yuv(FILE *outfile, struct yuv8_video_struct *yuv8_video);
int read_yuv(FILE *infile, struct yuv8_video_struct *yuv8_video);
```

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

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

**Note:** All image/video manipulation utility functions expect both input and output structures initialized; for example, pointing to a structure that has been allocated in memory, either as static or dynamic variables. Moreover, the input structure must have the dynamically allocated container (data or y, u, v) 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.

**Working with Video_struct Containers**

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

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

The `video_planes_per_mode` function returns the number of component planes defined by the mode variable, as described in Table 4-7. The `video_rows_per_plane` and `video_cols_per_plane` functions return the number of rows and columns in a given plane of the selected video structure. The following example demonstrates using these functions in conjunction to process all pixels within a video stream stored in the `in_video` variable:

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

**C-Model Example Code**

The example C demonstrator provided with the core, `run_bitacc_cmodel.c` demonstrates the steps required to run the C-model, by:

- Opening an example yuv file
- Increasing the color-component width to 10, 12, or 16 bits as necessary by shifting 8 bit data derived from the yuv file and padding the LSBs with X-Y ramp bits.
- Running the YCrCb2RGB C-model

After following the compilation instructions, run the example executable. The executable takes the path/name of the input file and the path of the output as parameters. If invoked with insufficient parameters, the following help message is printed:

```
Usage: run_bitacc_cmodel in_file out_path
       in_file : path/name of the input YUV file
       out_path : path to the output files
```

During successful execution, two directories will be created at the location specified by the `out_path` command line parameter. The first directory is the "expected" directory.
directory will contain a BMP file that corresponds to the output of the first frame that was processed. This directory will also contain a txt file called golden_1.txt. This txt file contains the output of the model in a format that can be directly used with the demonstration test bench. The second directory that is created is the "stimuli" directory. This directory will contain a txt file called stimuli_1.txt. This txt file contains the input of the model in a format that can be directly used with the demonstration test bench.

---

**Compiling with the YCrCb to RGB C-Model**

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

To compile the example code, first ensure that the directory in which the files `libIp_v_ycrcb2rgb_v7_0_bitacc_cmodel.so` and `libstlport.so.5.1` are located is present in your `$LD_LIBRARY_PATH` environment variable. These shared libraries are referenced during the compilation and linking process. Then cd into the directory where the header files, library files and `run_bitacc_cmodel.c` were unpacked. The libraries and header files are referenced during the compilation and linking process.

Place the header file and C source file in a single directory. Then in that directory, compile using the GNU C Compiler:

```bash
gcc -m32 -x c++ ../run_bitacc_cmodel.c ../gen_stim.c -o run_bitacc_cmodel -L. -lIp_v_ycrcb2rgb_v7_0_bitacc_cmodel -Wl,-rpath,.
gcc  -m64 -x c++ ../run_bitacc_cmodel.c ../gen_stim.c -o run_bitacc_cmodel -L. -lIp_v_ycrcb2rgb_v7_0_bitacc_cmodel -Wl,-rpath,.
```

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

Precompiled library `v_ycrcb2rgb_v7_0_bitacc_cmodel.dll`, and top level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. Here an example is presented using Microsoft Visual Studio.

In Visual Studio create a new, empty Windows Console Application project. As existing items, add:

- The `libIP_v_ycrcb2rgb_v7_0_bitacc_cmodel.dll` file to the "Resource Files" folder of the project
- The `run_bitacc_cmodel.c` and `gen_stim.c` files to the "Source Files" folder of the project
- The `v_ycrcb2rgb_v7_0_bitacc_cmodel.h` header files to "Header Files" folder of the project (optional)
After the project has been created and populated, it needs to be compiled and linked (built) to create a win32 executable. To perform the build step, choose **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.
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.

Vivado Integrated Design Environment (IDE)

The main screen of the Graphical User Interface (GUI) of Vivado IP Catalog shown in Figure 5-1 allows quick implementation of standard YCrCb to RGB or YUV to RGB converters without having to manually enter values from Table 3-2, Table 3-3 and Table 3-4. The Color-Space Converter core also supports proprietary (non-standard) converter implementations, by selecting “custom” from the Standard Selection drop-down menu, as long as the custom conversion matrix can be transformed to the form of Equation 3-3. Descriptions of the options provided in the GUI screens are included in this section.
The first page of the GUI displays the following options:

- **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_ycrcb2rgb_v7_0 cannot be used as a component name.

- **Video Component Width**: Specifies the bit width of input samples. Permitted values are 8, 10, 12 and 16 bits.

- **Pixels per Scanline (Default)**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the GUI as the default value for the lower half-word of the 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.

- **Scanlines per Frame (Default)**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the GUI as the default value for the upper half-word of the ACTIVE_SIZE register. When an AXI4-Lite interface is not present, the
GUI selection permanently defines the vertical size (number of lines) of the frames the generated core instance is to process.

**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 Debug Features:** When selected, the core will be generated with debugging features, which simplify system design, testing and debugging. For more information, refer to Debugging Features in Appendix C.

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

- **Enable INTC Port:** 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.

**Converter Type**

- **Standard Selection:** Select the standard to be implemented. The offered standards are:
  - YCrCb *ITU 601* (SD)
  - YCrCb *ITU 709* (HD) 1125/60 (PAL)
  - YCrCb *ITU 709* (HD) 1250/50 (NTSC)
  - YUV
  - custom

Selecting “custom” enables the controls on page 2 of the GUI, so conversion settings can be customized. Otherwise, page 2 only displays the parameters to implement the selected standard.

- **Output Range Selection:** This selection governs the range of outputs R, G and B by affecting the conversion coefficients as well as the clipping and clamping values. The core supports typical output ranges:
  - 16 to 235, typical for studio equipment
  - 16 to 240, typical for broadcast or television
  - 0 to 255, typical for computer graphics

The previously-mentioned ranges are characteristic for 8-bit outputs. If 10-, 12- or 16-bit outputs are used, the ranges are extended proportionally. For example, 16 to 240 mode for 10-bit outputs will result in output values ranging from 64 to 960.
The Conversion Matrix, Offset Compensation, Output Clipped and Output Clamped screen displays and enables editing of conversion coefficients, similar to Equation 3-3. Contents are editable only when “custom” is selected as the standard on page 1 (Figure 5-2).

• **Conversion Matrix**: Enter floating-point conversion constants, ranging from 0 to 1, into the four fields representing CA, CB, CC and CD.

• **Offset Compensation**: Enter the offset compensation constants (OY, OCb, and OCr in Equation 3-17, Equation 3-18 and Equation 3-19). These constants are scaled to the output representation. If OY, OCb, and OCr are in the 0.0 - 1.0 range, and the output is represented as 10-bit unsigned integers, then luminance and chrominance offsets should be entered as integers in the 0 - 1023 range.

• **Outputs Clipped/Outputs Clamped**: These check boxes control whether clipping/clamping logic will be instantiated in the generated netlist. The clipping/clamping logic ensures no arithmetic wrap-arounds happen at overflows, at the expense of extra slice-based logic resources.

---

Figure 5-2: Conversion Matrix, Offset Compensation, Clipping and Clamping Screen
• **Minimum and Maximum Values:** Similar to offset values, the edit boxes take unsigned integer values in the range permitted by the current output representation.

---

**Output Generation**

Vivado generates the files necessary to build the core and places those files in the `<project>/project.srcs/sources_1/ip/<core>` directory.

**File Details**

The Vivado output consists of some or all the following files:

*Table 5-1:*

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_rgb2ycrcb_v7_0</td>
<td>Library directory for the v_ycrcb2rgb_v7_0 core which contains the encrypted source files</td>
</tr>
<tr>
<td>v_tc_v6_0</td>
<td>Library directory for the helper core which contains the encrypted source files used with the v_ycrcb2rgb_v7_0</td>
</tr>
<tr>
<td>&lt;component_name&gt;.veo</td>
<td>The HDL template for instantiating the core.</td>
</tr>
<tr>
<td>&lt;component_name&gt;.vho</td>
<td></td>
</tr>
<tr>
<td>&lt;component_name&gt;.xci</td>
<td>IP-XACT file describing which options were used to generate the core. An XCI file can also be used as a source file for Vivado</td>
</tr>
<tr>
<td>&lt;component_name&gt;.xml</td>
<td>IP-XACT XML file describing how the core is constructed so Vivado can properly build the core.</td>
</tr>
</tbody>
</table>
Chapter 6

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.

Clock Frequencies

The pixel clock (ACLK) frequency is the required frequency for this core. See Maximum Frequencies in Chapter 2. 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 YCrCb to RGB Color-Space Converter v7.0 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 Suite. You are encouraged to make simple modifications to the configurations and observe the changes in the waveform.

Generating the Test Bench

1. 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.

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

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

\[<\text{PROJ\_DIR}>/\text{PROJ\_NAME}.srcs/sources_1/ip/\text{IP\_INSTANCE\_NAME}/demo_tb/\]
Directory and File Contents

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

- axi4lite_mst.v
- axi4s_video_mst.v
- axi4s_video_slv.v
- ce_generator.v
- tb_<IP_instance_name>.v
Test Bench Structure

The top-level entity is **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, see *Chapter 4, C Model Reference*.

- **axi4s_video_slv**
  The AXI4-Stream slave module, which acts as a passive slave to provide handshake signals for the AXI4-Stream transactions from the core output, can be used to open the data files generated from the reference C-model and verify the output from the core.

  To do this, edit **tb_<IP_instance_name>.v**:
  
  a. Add define macro for the golden file name and directory path
     ```
     define GOLDEN_FILE_NAME "<path><filename>"
     ```
  
  b. Comment-out the following line:
     ```
     SLV.is_passive;
     ```
     and replace with the following line:
     ```
     SLV.use_file(`GOLDEN_FILE_NAME);
     ```
For information on how to generate golden files, see Chapter 4, C Model Reference.

- ce_gen

Programmable Clock Enable (ACLKEN) generator.

**Running the Simulation**

There are two ways to run the demonstration test bench.

**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.

- Click **Simulation Settings** in the Flow Navigation window, change Simulation top module name to `tb_<IP_instance_name>`.
- Click **Run Simulation**. Vivado Simulator launches and you should be able to see the signals.
- You can also choose Questa SIM for simulation by going to **Project Settings** and selecting Questa SIM as the Target Simulator (Figure 7-3).
Option 2: Manually Compile and Run Simulation from Your Simulation Environment

- 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, see Generating the Test Bench.
- Setup the environment variables for Xilinx libraries.
- Compile the generated IP.
- Compile the test bench files.
- Run the simulation.

RECOMMENDED: Change the default simulation time from 1000 ns to all to be able observe a full frame transaction.
Verification, Compliance, and Interoperability

Simulation

A highly parameterizable test bench was used to test the YCrCb to RGB Color-Space Converter 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 $\text{ACLKEN}$ and $\text{ARESETn}$ assertion scenarios
• Testing of various frame sizes
• Varying parameter settings

Hardware Testing

The YCrCb to RGB Color-Space Converter 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 YCrCb to RGB Color-Space Converter 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 Video core which produces YCrCb (or YUV) 4:4:4 video data on an AXI4-Stream interface with a Video Protocol. The core master (output) RGB interface can work directly with any Video core which consumes RGB data on an AXI4-Stream interface with a Video Protocol.

- Reporting the Pass/Fail status of the test and any errors that were found
Migrating

For information about migration from ISE Design Suite to Vivado Design Suite, see *Vivado Design Suite Migration Methodology Guide* (UG911) [Ref 8].

For a complete list of Vivado User and Methodology Guides, see the Vivado Design Suite User Guides [Ref 9].

The core previously had aclk, aclken, and aresetn to control the Video over AXI4-Stream and AXI4-Lite interfaces. Currently, separate clock, clock enable, and reset pins control the Video over AXI4-Stream and the AXI4-Lite interfaces with clock domain crossing logic added to the core to handle the heterogeneous clock domains between the AXI4-Lite and Video over AXI4-Stream domains.

From version v6.01.a to v7.0 of the YCrCb2RGB core the following significant changes took place:

- The core is for native Vivado design tools release.
- This is a licensed core that you can acquire at no cost.
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 YCrCB to RGB Color-Space Converter 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 YCrCB to RGB Color-Space Converter core, 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 YCrCB to RGB Color-Space Converter core. 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.
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.

Answer Records for the YCrCB to RGB Color-Space Converter Core

AR 54542
http://www.xilinx.com/support/answers/54542.htm

Contacting Technical Support

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

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.

## Debug Tools

There are many tools available to address YCrCB to RGB Color-Space Converter core design issues. It is important to know which tools are useful for debugging various situations.

### Example Design

The YCrCB to RGB Color-Space Converter core is delivered with an example test bench. Information about the example test bench can be found in *Chapter 6, Example Design for the Vivado™ Design Suite*.

### Core Wizard

The YCrCB to RGB Color-Space Converter core is equipped with optional debugging features which aim to accelerate system bring-up, optimize memory and datapath 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. See the individual sections Core Bypass Option, Built-in Test-Pattern Generator, Throughput Monitors, and Evaluation Core Timeout for a description of these debug features.

### Vivado Lab Tools

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

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

• ILA 2.0 (and later versions)
• VIO 2.0 (and later versions)
Reference Boards

Various Xilinx development boards support YCrCB to RGB Color-Space Converter. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series evaluation boards
  - KC705
  - KC724

C Model Reference

See Chapter 4, C Model Reference 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 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:

- 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 Vivado Lab Tools are a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado Lab Tools for debugging the specific problems.

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
- Evaluation Core Timeout

General Checks

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

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

Core Bypass Option

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

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

In bypass mode the core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output.
Starting a system with all processing cores set to bypass, then by turning bypass off from
the system input towards the system output allows verification of subsequent cores with
known good stimuli.

**Built-in Test-Pattern Generator**

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

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

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

**Throughput Monitors**

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

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

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

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

The `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.

## Evaluation Core Timeout

The YCrCB to RGB Color-Space Converter hardware evaluation core times out after approximately eight hours of operation. The output is driven to zero. This results in a black screen for RGB color systems and in a dark-green screen for YUV color systems.

## Interface Debug

### AXI4-Lite Interfaces

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

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

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

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

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

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0 of the <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 (<strong>EOL</strong>) 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 Vivado Lab Tools, measure the number of active AXI4-Stream transactions between <strong>EOL</strong> 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 (<strong>EOL</strong>) 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 Vivado Lab Tools, measure the number of active AXI4-Stream transactions between <strong>EOL</strong> pulses.</td>
</tr>
<tr>
<td>Bit 2 or Bit 3 of the <strong>ERROR</strong> register reads back set.</td>
<td>Bit 2 of the <strong>ERROR</strong> register, <strong>SOF_EARLY</strong>, and bit 3 of the <strong>ERROR</strong> register <strong>SOF_LATE</strong> indicate the number of pixels received between the latest and the preceding Start-Of-Frame (<strong>SOF</strong>) differ from 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 Vivado Lab Tools, measure the number of active AXI4-Stream transactions between subsequent <strong>SOF</strong> pulses.</td>
</tr>
<tr>
<td><strong>s_axis_video_tready</strong> stuck low, the upstream core cannot send data.</td>
<td>During initialization, line-, and frame-flushing, the core keeps its <strong>s_axis_video_tready</strong> input low. Afterwards, the core should assert <strong>s_axis_video_tready</strong> automatically. Is <strong>m_axis_video_tready</strong> low? If so, the core cannot send data downstream, and the internal FIFOs are full.</td>
</tr>
</tbody>
</table>
| **m_axis_video_tvalid** stuck low, the downstream core is not receiving data | • No data is generated during the first two lines of processing.  
• If the programmed active number of pixels per line is radically smaller than the actual line length, the core drops most of the pixels waiting for the (**s_axis_video_tlast**) End-of-line signal. Check the **ERROR** register. |
| Generated **SOF** signal (**m_axis_video_tuser0**) signal misplaced. | Check the **ERROR** register. |
| Generated **EOL** signal (**m_axis_video_tlast**) signal misplaced. | Check the **ERROR** register. |
Appendix C: Debugging

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

Other Interfaces

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

<table>
<thead>
<tr>
<th>Table C-2: Troubleshooting AXI4-Stream Interface (Cont’d)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptom</strong></td>
</tr>
<tr>
<td>Data samples lost between Upstream core and this core. Inconsistent EOL and/or SOF periods received.</td>
</tr>
<tr>
<td>Data samples lost between Downstream core and this core. Inconsistent EOL and/or SOF periods received.</td>
</tr>
</tbody>
</table>

Table C-3: Troubleshooting Third-Party Interfaces

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

Programmer Guide

The software API is provided to allow easy access to the YCrCb2RGB AXI4-Lite registers defined in Table 2-8. To utilize the API functions, the following two header files must be included in the user C code:

```c
#include "ycrcb2rgb.h"
#include "xparameters.h"
```

The hardware settings of your system, including the base address of your YCrCb2RGB core, are defined in the `xparameters.h` file. The `ycrcb2rgb.h` file contains the macro function definitions for controlling the YCrCb2RGB 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 `ycrcb2rgb_v7_00_a/example` subfolder. This file is a sample C program that demonstrates how to use the YCrCb2RGB pCore API.

Table D-1: YCrCb2RGB Driver Function Definitions

<table>
<thead>
<tr>
<th>Function Name and Parameterization</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YCC_Enable (uint32 BaseAddress)</td>
<td>Enables a YCrCb2RGB instance.</td>
</tr>
<tr>
<td>YCC_Disable (uint32 BaseAddress)</td>
<td>Disables a YCrCb2RGB instance.</td>
</tr>
<tr>
<td>YCC_Reset (uint32 BaseAddress)</td>
<td>Immediately resets a YCrCb2RGB instance. The core stays in reset until the RESET flag is cleared.</td>
</tr>
<tr>
<td>YCC_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>YCC_AutoSyncReset (uint32 BaseAddress)</td>
<td>Resets a YCrCb2RGB 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>YCC_ReadReg (uint32 BaseAddress, uint32 RegOffset)</td>
<td>Returns the 32-bit unsigned integer value of the register. Read the register selected by RegOffset (defined in Table 2-8).</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. YCC_Reset() and YCC_AutoSyncReset() reset the core immediately if the core is not currently processing a frame. If the core is currently processing a frame calling YCC_Reset(), or setting bit 30 of the CONTROL register to 1 will cause image tearing. After calling YCC_Reset(), the core remains in reset until YCC_ClearReset() is called.

Calling YCC_AutoSyncReset() 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 YCC_AutoSyncReset(), it is not necessary to call YCC_ClearReset() for the core to return to normal operating mode.

<table>
<thead>
<tr>
<th>Function Name and Parameterization</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YCC_WriteReg</td>
<td>Write the register selected by RegOffset (defined in Table 2-8. Data is the 32-bit value to write to the register.</td>
</tr>
<tr>
<td>YCC_RegUpdateEnable</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>YCC_RegUpdateDisable</td>
<td>Disables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.</td>
</tr>
<tr>
<td>YCC_select_standard</td>
<td>Populates an rgb_coef_inputs structure with the values from the selected Video standard standard_sel 0 = SD_ITU_601 1 = HD_ITU_709_1125_NTSC 2 = HD_ITU_709_1250_PAL 3 = YUV input_range 0 = 16_to_240_for_TV, 1 = 16_to_235_for_Studio_Equipment 2 = 0_to_255_for_Computer_Graphics</td>
</tr>
<tr>
<td>YCC_coefficient_translation</td>
<td>Translates the ycc_coef_inputs structure into the ycc_coef_outputs structure that can be used to program the core's registers. The ycc_coef_inputs structure uses the same values as the core's GUIs. The ycc_coef_outputs structure uses the values that can be programmed into the core's registers.</td>
</tr>
<tr>
<td>void YCC_set_coefficients</td>
<td>Writes the translated coefficient values to the core's registers.</td>
</tr>
<tr>
<td>void YCC_get_coefficients</td>
<td>Reads the translated coefficient values from the core's registers.</td>
</tr>
</tbody>
</table>

Table D-1: YCrCb2RGB Driver Function Definitions (Cont’d)
**IMPORTANT:** Calling `YCC_AutoSyncReset()` 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

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

If multiple register values are changed during frame processing, simple double buffering would not guarantee that all register updates would take effect at the beginning of the same frame. Using a semaphore mechanism, the `RegUpdateEnable()` and `RegUpdateDisable()` functions allows synchronous commitment of register changes. The YCrCb2RGB core will start using the updated ACTIVE_SIZE and register values only if the REGUPDATE flag of the CONTROL register is set (1), after the next Start-Of-Frame signal (s_axis_video_tuser0) 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 2-8 has a constant that is defined in ycrcb2rgb.h which is set to the offset for that register listed in Table D-2. It is recommended that the application software uses the predefined register names instead of register values when accessing core registers, so future updates to the YCrCb2RGB drivers which may change register locations will not affect the application dependent on the YCrCb2RGB driver.

<table>
<thead>
<tr>
<th>Constant Name Definition</th>
<th>Value</th>
<th>Target Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>YCC_CONTROL</td>
<td>0x0000</td>
<td>CONTROL</td>
</tr>
<tr>
<td>YCC_STATUS</td>
<td>0x0004</td>
<td>STATUS</td>
</tr>
<tr>
<td>YCC_ERROR</td>
<td>0x0008</td>
<td>ERROR</td>
</tr>
<tr>
<td>YCC_IRQ_ENABLE</td>
<td>0x000C</td>
<td>IRQ_ENABLE</td>
</tr>
<tr>
<td>YCC_VERSION</td>
<td>0x0010</td>
<td>VERSION</td>
</tr>
<tr>
<td>YCC_SYSDEBUG0</td>
<td>0x0014</td>
<td>SYSDEBUG0</td>
</tr>
<tr>
<td>YCC_SYSDEBUG1</td>
<td>0x0018</td>
<td>SYSDEBUG1</td>
</tr>
</tbody>
</table>
### Table D-2: Predefined Constants Defined in ycrb2rgb.h (Cont’d)

<table>
<thead>
<tr>
<th>Constant Name Definition</th>
<th>Value</th>
<th>Target Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>YCC_SYSDEBUG2</td>
<td>0x001C</td>
<td>SYSDEBUG2</td>
</tr>
<tr>
<td>YCC_ACTIVE_SIZE</td>
<td>0x0020</td>
<td>ACTIVE_SIZE</td>
</tr>
<tr>
<td>YCC_RGBMAX</td>
<td>0x100</td>
<td>YMAX</td>
</tr>
<tr>
<td>YCC_RGBMIN</td>
<td>0x104</td>
<td>YMIN</td>
</tr>
<tr>
<td>YCC_ROFFSET</td>
<td>0x108</td>
<td>YOFFSET</td>
</tr>
<tr>
<td>YCC_GOFFSET</td>
<td>0x10C</td>
<td>CBOFFSET</td>
</tr>
<tr>
<td>YCC_BOFFSET</td>
<td>0x110</td>
<td>CROFFSET</td>
</tr>
<tr>
<td>YCC_ACOEF</td>
<td>0x114</td>
<td>ACOEF</td>
</tr>
<tr>
<td>YCC_BCOEF</td>
<td>0x118</td>
<td>BCOEF</td>
</tr>
<tr>
<td>YCC_CCOEF</td>
<td>0x11C</td>
<td>CCOEF</td>
</tr>
<tr>
<td>YCC_DCOEF</td>
<td>0x120</td>
<td>DCOEF</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:

http://www.xilinx.com/support

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:

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

References

These documents provide supplemental material useful with this user guide:

7. AXI Reference Guide (UG761)
8. *Vivado Design Suite Migration Methodology Guide* ([UG911](#))
9. *Vivado Design Suite User Documentation*

---

**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 DS659 and UG833.</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. Added Vivado test bench.</td>
</tr>
<tr>
<td>03/20/2012</td>
<td>4.0</td>
<td>Updated for core version. Updated Debugging appendix. Removed ISE chapter.</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 2011–2013 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.