

# LogiCORE IP Gamma Correction v4.0

## *Product Guide*

PG004 October 19, 2011

# Table of Contents

---

## Chapter 1: Overview

Standards Compliance .....	6
Feature Summary .....	6
Applications .....	6
Licensing .....	6
Performance .....	7
Resource Utilization.....	8
Optimization Options.....	10

## Chapter 2: Core Interfaces

Core Symbol and Port Descriptions.....	13
--	----

## Chapter 3: Customizing and Generating the Core

Graphical User Interface (GUI) .....	23
Parameter Values in the XCO File .....	25
Output Generation.....	25

## Chapter 4: Constraining the Core

Required Constraints.....	28
Device, Package, and Speed Grade Selections.....	28
Clock Frequencies.....	28
Clock Management .....	28
Clock Placement .....	28
Banking.....	28
Transceiver Placement .....	28
I/O Standard and Placement.....	28

## Chapter 5: Designing with the Core

Control Signals and Timing.....	29
Clocking.....	29
Resets.....	29
Protocol Description .....	29

## Appendix A: Verification, Compliance, and Interoperability

Simulation.....	30
Hardware Testing .....	30

---

<b>Appendix B: Migrating</b>	
Special Considerations when Migrating to AXI .....	31
<b>Appendix C: Debugging</b>	
Evaluation Core Timeout .....	32
<b>Appendix D: Application Software Development</b>	
EDK pCore API Functions .....	33
<b>Appendix E: C Model Reference</b>	
Gamma Correction Input and Output Video Structure .....	36
Initializing the Gamma Correction Input Video Structure .....	37
Example Code .....	39
Compiling Gamma Correction C Model with Example Wrapper .....	39
<b>Appendix F: Additional Resources</b>	
Xilinx Resources .....	41
Solution Centers .....	41
References .....	41
Technical Support .....	41
Ordering Information .....	42
Revision History .....	42
Notice of Disclaimer .....	42

## Introduction

The Xilinx Gamma Correction LogiCORE™ provides customers with an optimized hardware block for manipulating image data to match the response of display devices. This core is implemented using a look-up table structure that is programmed to implement a gamma correction curve transform on the input image data. Programmable number of Gamma tables enable having separate gamma tables for all color channels, separate tables for luminance and chrominance channels, or one gamma table to be shared by all color channels.

CORE Generator™ technology generates the core as either an AXI EDK pCore, a standalone netlist for a General Purpose Processor (GPP), or as a Fixed Mode netlist. When generated as an EDK pCore, the processor interface is AXI4-Lite compliant.

## Features

- Programmable gamma tables
- Single or three color channel independent or shared look-up table structure
- Optional interpolated output values
- Selectable processor interface
  - EDK pCore  
AXI interface based on AXI4-Lite specification
  - General Purpose Processor
  - Constant Interface
- Optional double-buffered interface to prevent image tearing
- 8-, 10-, or, 12-bit input and output precision
- Delay match support for up to three sync signals
  - For use with Xilinx CORE Generator™ software v13.3 or higher

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family <sup>(1)</sup>	Virtex®-7, Kintex®-7, Virtex®-6, Spartan®-6
Supported User Interfaces	General Processor Interface, EDK pCore AXI4-Lite, Constant Interface
Resources	See <a href="#">Table 1-1</a> through <a href="#">Table 1-4</a> .
Provided with Core	
Documentation	Product Specification
Design Files	Netlists, EDK pCore files
Example Design	Not Provided
Test Bench	VHDL <sup>(3)</sup>
Constraints File	Not Provided
Simulation Models	VHDL or Verilog Structural, C Model <sup>(3)</sup>
Tested Design Tools	
Design Entry Tools	CORE Generator tool, Platform Studio (XPS)
Simulation <sup>(2)</sup>	Mentor Graphics ModelSim, Xilinx® ISim 13.3
Synthesis Tools	ISE 13.3
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the [release notes](#) for this core.
2. HDL test bench and C Model available on the product page on Xilinx.com at <http://www.xilinx.com/products/ipcenter/EF-DI-GAMMA.htm>
3. For the supported versions of the tools, see the [ISE Design Suite 13: Release Notes Guide](#).

# Overview

---

Gamma correction, also known as gamma compression or encoding, is used to encode linear luminance or RGB values to match the non-linear characteristics of display devices. Gamma correction helps to map data into a more perceptually uniform domain, so as to optimize perceptual performance of a limited signal range, such as a limited number of bits in each RGB component.

Gamma correction is, in the simplest cases, defined by

$$V_{out} = V_{in}^{\gamma}$$

where the input and output values are between 0 and 1 (Figure 1-1). The case  $\gamma < 1$  is often called gamma compression and  $\gamma > 1$  is called gamma expansion.

When used in conjunction with an embedded or external processor, the Gamma Correction core supports frame-by-frame dynamic reprogramming of the gamma tables. The gamma tables can be reprogrammed with arbitrary functions, supporting a wide range of applications, such as intensity correction, feature enhancement, lin-log, log-lin conversion and thresholding.

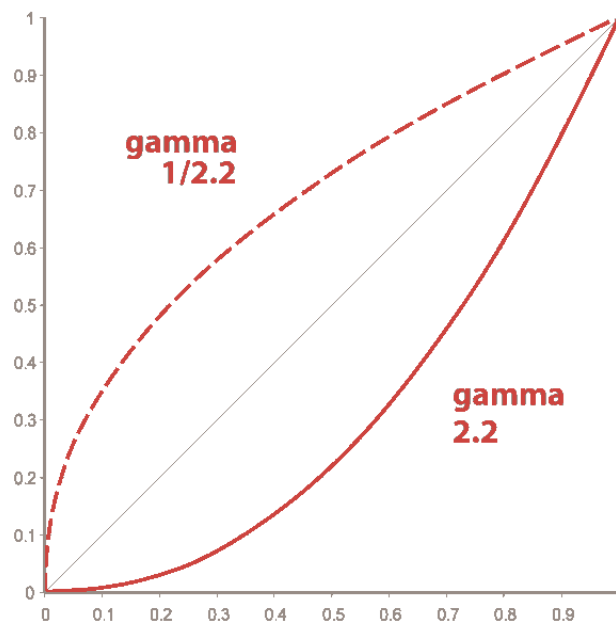


Figure 1-1: Gamma Correction

The Gamma Correction core also offers various configuration options for a designer to optimize the block RAM footprint required by the core.

## Standards Compliance

The Gamma Correction core is compliant with the AXI4-Lite interconnect standard as defined in the *AXI Reference Guide (UG761)*.

## Feature Summary

The Gamma Correction core provides programmable look-up tables for gamma correction. A programmable number of Gamma tables allows for separate gamma tables for all color channels, separate tables for luminance and chrominance channels, or one gamma table to be shared by all color channels.

## Applications

- Pre-processing block for image sensors
- Post-processing block for image data adjustment
- Intensity correction
- Video surveillance
- Consumer displays
- Video conferencing
- Machine vision

## Licensing

The Gamma Correction core provides the following three licensing options:

- Simulation Only
- Full System Hardware Evaluation
- Full

After installing the required Xilinx ISE software and IP Service Packs, choose a license option.

### Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess core functionality with either the example design provided with the Gamma Correction core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

### Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the Gamma Correction core using the example design and demonstration test bench provided with the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

To obtain a Full System Hardware Evaluation license, do the following:

1. Navigate to the [product page](#) for this core.
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

## Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Full implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time outs

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

## Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

## Performance

### Maximum Frequencies

The following are typical clock frequencies for the target devices. The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the Field Programmable Gate Array (FPGA) device, using a different version of Xilinx tools, and other factors.

- Virtex®-7 FPGA: 250 MHz
- Kintex™-7 FPGA: 250 MHz
- Virtex-6 FPGA: 250 MHz
- Spartan®-6 FPGA: 150 MHz

### Latency

The propagation delay of the Gamma Correction core is always four clock cycles.

## Resource Utilization

For an accurate measure of the usage of device resources (for example, block RAMs, flip-flops, and look-up tables) for a particular instance, click **View Resource Utilization** in CORE Generator after generating the core.

The information presented in [Table 1-1](#) through [Table 1-4](#) is a guide to the resource utilization of the Color Correction Matrix core for all input/output width combinations for Virtex-7, Kintex-7, Virtex-6, and Spartan-6 FPGA families. This core does not use any DSP48s, dedicated I/O, or CLK resources. The design was tested using ISE® v13.3 tools with default tool options for characterization data.

Table 1-1: Virtex-7 xc7vx330t,ffg1761,C,-1

Input Width	Output Width	Interpolation	LUT6-FF Pairs	LUTs	FFs	RAM16/8	Clock Frequency (MHz)
8	8	No	184	51	183	0/ 3	397
8	10	No	181	66	195	0/ 3	397
8	12	No	213	45	207	0/ 3	370
10	8	No	180	66	195	0/ 3	397
10	10	No	206	51	207	0/ 3	397
10	12	No	208	63	219	0/ 3	370
12	8	No	205	53	207	3/ 0	397
12	10	No	212	58	219	3/ 3	370
12	12	No	230	53	231	3/ 3	397
12	8	Yes	330	187	219	0/ 3	264
12	10	Yes	362	212	231	0/ 3	264
12	12	Yes	391	234	243	0/ 3	274

1. Speedfile: ADVANCED 1.01 2011-09-26

Table 1-2: Kintex-7 xc7k70t,fbg676,C,-1

Input Width	Output Width	Interpolation	LUT6-FF Pairs	LUTs	FFs	RAM16/8	Clock Frequency (MHz)
8	8	No	124	104	183	0/ 3	398
8	10	No	124	110	195	0/ 3	398
8	12	No	183	76	207	0/ 3	398
10	8	No	163	76	195	0/ 3	398
10	10	No	208	50	207	0/ 3	398
10	12	No	207	63	219	0/ 3	398
12	8	No	163	84	207	3/ 0	398
12	10	No	206	57	219	3/ 3	398
12	12	No	182	99	231	3/ 3	398

Table 1-2: Kintex-7 xc7k70t,fbg676,C,-1 (Cont'd)

12	8	Yes	329	192	219	0/ 3	275
12	10	Yes	363	208	231	0/ 3	295
12	12	Yes	374	238	243	0/ 3	255

1. Speedfile: ADVANCED 1.02 2011-09-26

Table 1-3: Virtex-6 xc6vlx75t,ff484,C,-1

Input Width	Output Width	Interpolation	LUT6-FF Pairs	LUTs	FFs	RAM16/8	Clock Frequency (MHz)
8	8	No	177	57	183	0/ 3	400
8	10	No	195	47	195	0/ 3	400
8	12	No	184	74	207	0/ 3	400
10	8	No	180	66	195	0/ 3	400
10	10	No	190	69	207	0/ 3	400
10	12	No	213	58	219	0/ 3	400
12	8	No	196	59	207	3/ 0	400
12	10	No	203	66	219	3/ 3	400
12	12	No	231	51	231	3/ 3	400
12	8	Yes	337	201	219	0/ 3	274
12	10	Yes	375	214	231	0/ 3	262
12	12	Yes	394	237	246	0/ 3	267

1. Speedfile: PRODUCTION 1.15 2011-09-26

Table 1-4: Spartan-6 xc6slx9,csg225,C,-2

Input Width	Output Width	Interpolation	LUT6-FF Pairs	LUTs	FFs	RAM16/8	Clock Frequency (MHz)
8	8	No	149	119	186	0/ 3	279
8	10	No	167	116	198	0/ 3	279
8	12	No	167	129	210	0/ 3	279
10	8	No	167	127	198	0/ 3	279
10	10	No	180	126	210	3/ 0	279
10	12	No	184	133	222	3/ 0	279
12	8	No	200	125	218	4/ 0	279
12	10	No	202	137	228	3/ 3	279
12	12	No	196	129	222	3/ 0	279

1. Speedfile: PRODUCTION 1.20 2011-09-26

## Optimization Options

Various configuration settings determine how many Block RAM primitives are used. For example, sharing correction curves between two or three color channels, using 8- or 10-bit input, and single-buffering requires only a single Block RAM primitive. Table 1-5 and Table 1-6 show Block RAM usage for a number of typical configurations.

Table 1-5: Block-RAM Footprint When Interpolation is Not Used

IWIDTH	OWIDTH	DBLBUF	Look-up* Tables	Spartan-6	Virtex-6	
				RAMB16	RAMB18	RAMB36
8	8	0	1,2	2	2	0
8	8	0	3	3	3	0
8	8	1	3	3	3	0
8	10	0	1,2	2	2	0
8	10	0	3	3	3	0
8	10	1	3	3	3	0
8	12	0	1,2	2	2	0
8	12	0	3	3	3	0
8	12	1	3	3	3	0
10	8	0	1,2	2	2	0
10	8	0	3	3	3	0
10	8	1	3	3	3	0
10	10	0	1,2	2	2	0
10	10	0	3	3	3	0
10	10	1	3	6	0	3
10	12	0	1,2	2	2	0
10	12	0	3	3	3	0
10	12	1	3	6	0	3
12	8	0	1,2	4	2	0
12	8	0	3	6	0	3
12	8	1	3	12	0	6
12	10	0	1,2	6	2	2
12	10	0	3	9	0	3
12	10	1	3	15	0	6
12	12	0	1,2	6	2	2
12	12	0	3	9	0	3
12	12	1	3	18	0	9

\* The number of lookup tables used is as follows:  
 3: when Independent look-up tables for each Color Channel is selected  
 2: when Identical look-up tables for Chrominance Channels Only is selected

1: when Identical look-up tables for all Color Channels is selected

Table 1-6: Block-RAM Footprint When Interpolation is Used

IWIDTH	OWIDTH	DBLBUF	Look-up Tables*	Spartan-6	Virtex-6	
				RAMB16	RAMB18	RAMB36
12	8	0	3	3	3	0
12	8	1	3	6	6	0
12	10	0	3	3	3	0
12	10	1	3	6	6	0
12	12	0	3	3	3	0
12	12	1	3	6	6	0

\* The number of lookup tables used is as follows:

- 3: when Independent look-up tables for each Color Channel is selected
- 2: when Identical look-up tables for Chrominance Channels Only is selected
- 1: when Identical look-up tables for all Color Channels is selected

## Shared Look-Up Tables

When multiple channels require the same correction curve, a single look-up table may be shared between two or more channels. Sharing look-up tables between multiple channels reduces the number of write operations required to update the look-up tables for EDK pCore and General Purpose Processor interfaces. It also can reduce the number of Block RAM resources used (see [Table 1-5](#)).

## Interpolation of Look-up Table Contents

When the gamma function is configured for 12-bit input data, an optional look-up table interpolation is provided to reduce the size of look-up tables and thereby the number of block RAMs. This interpolation stores every 4th sample in the look-up table ([Figure 1-2](#)), which can reduce the number of block RAMs used by 75%.

The Gamma Correction core supports linear interpolation, which trades off block RAM(s) for adders to implement the 1-to-4 interpolation. When used to interpolate sufficiently smooth functions, such as the power functions used for gamma correction, the interpolation error is orders of magnitude smaller than the output quantization error.

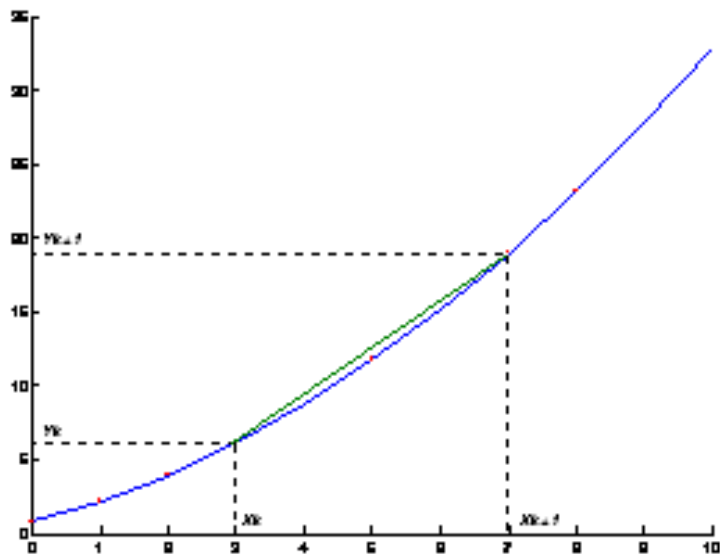


Figure 1-2: Interpolation of Look-up Table Contents

## Core Interfaces

---

### Core Symbol and Port Descriptions

As discussed previously, the Gamma Correction core can be configured with three different interface options, each resulting in a slightly different set of ports. The Gamma Correction IP core uses a set of signals that is common to all of the Xilinx Video IP cores called the Xilinx Streaming Video Interface (XSVI). The XSVI signals are common to all interface options and are shown in [Figure 2-1](#) and described in [Table 2-1](#).

#### Xilinx Streaming Video Interface

The Xilinx Streaming Video Interface (XSVI) is a set of signals common to all of the Xilinx video cores used to stream video data between IP cores. XSVI is also defined as an Embedded Development Kit (EDK) bus type so that the tool can automatically create input and output connections to the core. This definition is embedded in the pCORE interface provided with the IP, and it allows an easy way to cascade connections of Xilinx Video Cores. The Gamma Correction IP core uses the following subset of the XSVI signals:

- video\_data
- vblank
- hblank
- active\_video

Other XSVI signals on the XSVI input bus, such as `video_clk`, `vsync`, `hsync`, `field_id`, and `active_chr` do not affect the function of this core.

**Note:** These signals are neither propagated, nor driven on the XSVI output of this core.

The following is an example EDK Microprocessor Peripheral Definition (.MPD) file definition. `IWIDTH` and `OWIDTH` are the values you selected when you generated the IP in CORE Generator (i.e., 8, 10, or 12).

Input Side:

```
BUS_INTERFACE BUS = XSVI_GAMMA_IN, BUS_STD = XSVI, BUS_TYPE = TARGET

PORT active_video_in = active_video, BUS = XSVI_GAMMA_IN, DIR = IN,
PORT hblank_in      = hblank,        BUS = XSVI_GAMMA_IN, DIR = IN
PORT vblank_in      = vblank,        BUS = XSVI_GAMMA_IN, DIR = IN
PORT video_data_in  = video_data,    VEC = [0:((IWIDTH*3)-1)], BUS=XSVI_GAMMA_IN, DIR = IN
```

Output Side:

```
BUS_INTERFACE BUS = XSVI_GAMMA_OUT, BUS_STD = XSVI, BUS_TYPE = INITIATOR,

PORT active_video_out = active_video, BUS = XSVI_GAMMA_OUT, DIR =OUT,
```

```

PORT hblank_out      = hblank,          BUS = XSVI_GAMMA_OUT  DIR = OUT
PORT vblank_out     = vblank,          BUS = XSVI_GAMMA_OUT, DIR = OUT,
PORT video_data_out = video_data, VEC = [0:((OWIDTH*3)-1)], BUS = XSVI_GAMMA_OUT, DIR=OUT,
    
```

The Gamma Correction IP core is fully synchronous to the core clock, `clk`. Consequently, the input XSVI bus is expected to be synchronous to the input clock, `clk`. Similarly, to avoid clock resampling issues, the output XSVI bus for this IP is synchronous to the core clock, `clk`. The `video_clk` signals of the input and output XSVI buses are not used.

## Constant Interface

As this interface does not provide additional programmability, the Constant Interface has no ports other than the Xilinx Streaming Video Interface, `clk`, `ce`, and `sclr` signals. The Constant Interface Core Symbol is shown in Figure 2-1.

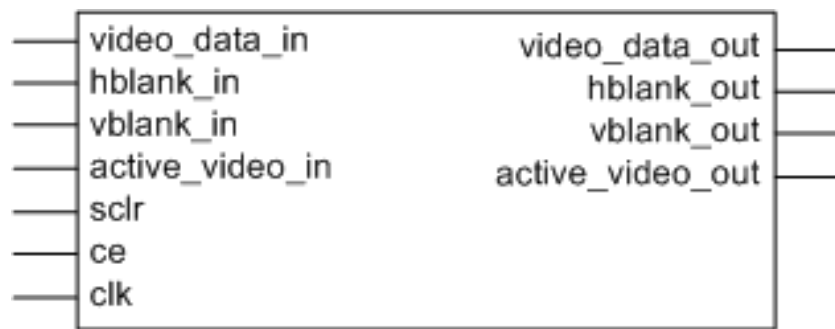


Figure 2-1: Core Symbol for the Constant Interface

Table 2-1: Port Descriptions for the Constant Interface

Port Name	Port Width	Direction	Description
video_data_in	3*IWIDTH	IN	Data input bus
hblank_in	1	IN	Horizontal blanking input
vblank_in	1	IN	Vertical blanking input
active_video_in	1	IN	Active video signal input
video_data_out	3*OWIDTH	OUT	Data output bus
hblank_out	1	OUT	Horizontal blanking output
vblank_out	1	OUT	Vertical blanking output
active_video_out	1	OUT	Active video signal output
clk	1	IN	Rising-edge clock
ce	1	IN	Clock enable (active high)
sclr	1	IN	Synchronous clear - reset (active high)

- video\_data\_in:** This bus contains the three individual color inputs in the following order from MSB to LSB [red : blue : green]. Color values are expected in IWIDTH bits

wide unsigned integer representation. The Gamma Correction core supports RGB, YUV, and YCrCb input formats.

Bits	3IWIDITH-1:2IWIDITH	2IWIDITH-1:IWIDITH	IWIDITH-1:0
Video Data Signals	Red/U/Cr	Blue/V/Cb	Green/Y

- **hblank\_in:** The `hblank_in` signal conveys information about the blank/non-blank regions of video scan lines. This signal is not actively used in the Gamma Correction core, but passed through the core with a delay matching the latency of the corrected data.
- **vblank\_in:** The `vblank_in` signal conveys information about the blank/non-blank regions of video frames, and is used by the Gamma Correction core to detect beginning of a frame, when user registers can be copied to active registers to avoid visual tearing of the image. This signal is passed through the core with a delay matching the latency of the corrected data.
- **active\_video\_in:** The `active_video_in` signal is high when valid data is presented at the input. This signal is not actively used in the Gamma Correction core, but passed through the core with a delay matching the latency of the corrected data.
- **clk - clock:** Master clock in the design, synchronous with, or identical to the video clock.
- **ce - clock enable:** Pulling `CE` low suspends all operations within the core. Outputs are held, and no input signals are sampled, except for reset (`SCLR` takes precedence over `CE`).
- **sclr - synchronous clear:** Pulling `SCLR` high results in resetting all output ports to zero or their default values. Internal registers within the XtremeDSP™ slice and D-flip-flops are cleared. However, the core uses `SRL16/SRL32` based delay lines for `hblank_out`, `vblank_out`, and `active_video_out` generation, which are not cleared by `SCLR`. This may result in non-zero outputs after `SCLR` is de-asserted, until the contents of `SRL16/SRL32s` are flushed. Unwanted results can be avoided if `SCLR` is held active for 16/32 clock cycles until `SRL16s/SRL32s` are flushed. Also, `SCLR` does not reset the contents of the gamma look-up tables within the core.
- **video\_data\_out:** This bus contains video output in the same order as `video_data_in`. Color values are represented as `OWIDTH` bits wide unsigned integers. The format of the output (RGB, YUV, or YCrCb) will match the input.

Bits	3*OWIDITH-1:2*OWIDITH	2*OWIDITH-1:OWIDITH	OWIDITH-1:0
Video Data Signals	Red/U/Cr	Blue/V/Cb	Green/Y

- **hblank\_out and vblank\_out:** The corresponding input signals are delayed so blanking outputs are in phase with the video data output, maintaining the integrity of the video stream. The blanking outputs are connected to the corresponding inputs via delay-lines matching the propagation delay of the video processing pipe. Unwanted blanking inputs should be tied high, and corresponding outputs left unconnected, which will result in the trimming of any unused logic within the core.
- **active\_video\_out:** The `active_video_out` signal is high when valid data is present at the output. The `active_video_out` signal is connected to `active_video_in` via delay-lines matching the propagation delay of the video processing pipe. The `active_video` signal does not affect the processing behavior of the core. Asserting or deasserting it will not stall processing or the video stream, nor will it force video outputs to zero.

## Single- and Double-Buffered EDK pCore Interfaces

The Single- and Double-Buffered EDK pCore Interfaces generate additional AXI4-Lite Bus interface ports in addition to the `clk`, `ce`, `sclr`, and XSVI signals. The AXI4-Lite bus signals are automatically connected when the generated pCore is inserted into an EDK project. The Core Symbol for the Single- and Double-Buffered EDK pCore Interfaces is shown in [Figure 2-2](#). The Xilinx Streaming Video Interface `clk`, `ce`, and `sclr`, signals are described in the previous section. For more information on the AXI4-Lite bus signals, see [AXI Reference Guide](#).

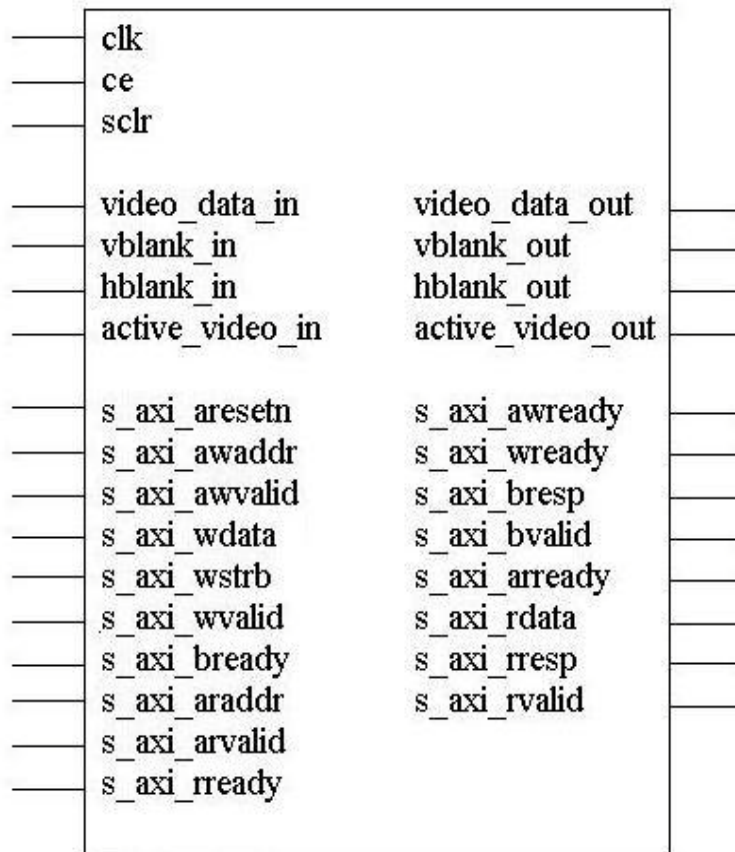


Figure 2-2: Core Symbol for the EDK pCore Interface

## EDK pCore Interface

Many imaging applications have an embedded processor that can dynamically control the parameters within the core. The EDK pCore Interface creates a pCore that can be added to an EDK project as a hardware peripheral. This pCore provides a memory-mapped interface for the programmable registers within the core, which are described in [Table 2-2](#).

Table 2-2: EDK pCore Interface Register Descriptions

Address Offset (hex)	Register Name	Access Type	Default Value	Description
BASEADDR + 0x00	gamma_reg0_control	R/W	0x00000001	Bit 0: Software Enable <ul style="list-style-type: none"> <li>• 0 – Not enabled</li> <li>• 1 – Enabled</li> </ul> Bit 1: Table Update When using Double Buffering, bit 1 is a control bit for triggering the update of the active look-up table on the next VBlank rising-edge. <ul style="list-style-type: none"> <li>• 0 – Normal Operating Mode, continue to operate with current look-up table.</li> <li>• 1 – look-up table update. The inactive look-up table with newly written data will become the active look-up table on the next VBlank rising edge.</li> </ul>
BASEADDR + 0x04	gamma_reg1_reset	R/W	0x00000000	Bit 0: Software Manual Reset <ul style="list-style-type: none"> <li>• 0 – Not Reset</li> <li>• 1 – Force immediate reset, hold until bit is cleared</li> </ul> Bit 1: Software Auto-synchronized Reset <ul style="list-style-type: none"> <li>• 0 – Not Reset</li> <li>• 1 – Reset will occur automatically on the next rising-edge of vblank_in</li> </ul>
BASEADDR + 0x08	gamma_reg2_status	R	0x00040000	Bits [15-0]: Not Used Bits [31-16]: DRIVER_PARAMS & DRIVER_VERSION
BASEADDR + 0x0C	gamma_reg3_addr_data	R/W	0x00000000	Bits [15-0]: Value to write to gamma look-up table Bits [31-16]: Target address in look-up table

All of the registers are readable, enabling you to verify writes or read back current values.

The Gamma Correction core has a feature that allows it to be enabled or disabled. This halts the operation of the core by blocking the propagation of all video signals. This function is controlled by setting the Software Enable, bit 0 of `gamma_reg0_control` register, to 0; the default value of Software Enable is 1 (enabled).

When using a single-buffered interface, write operations to the `gamma_reg3_data` port take effect immediately on the active look-up tables in the Gamma Correction core, and may cause image tearing if updated when the `active_video` signal = '1'.

When using a double-buffered interface, write operations to the `gamma_reg3_data` port take effect in a secondary inactive look-up table, and can occur at any time safely. When write operations are complete, assert `gamma_reg0_control` bit 1 to automatically swap the active look-up table with the inactive look-up table at the next rising edge of VBlank, which will cause the following frame to be processed using the newly updated gamma function.

The Software Reset register `gamma_reg1_reset` can be used to reset the state of the Gamma Correction core. Setting bit 0 of `gamma_reg1_reset` to 1 resets the core's state

immediately, while setting bit 1 of `gamma_reg1_reset` to 1 causes a reset to occur automatically on the next rising edge of VBlank. In both cases, reset does not alter the contents of the gamma look-up tables, which can only be updated by explicitly writing new contents to the core.

Pin Name	Dir	Width	Description
<b>AXI Global System Signals(1)</b>			
S_AXI_ARESETN	I	1	AXI Reset, active low
IP2INTC_Irpt	O	1	Interrupt request output
<b>AXI Write Address Channel Signals(1)</b>			
S_AXI_AWADDR	I	[(C_S_AXI_ADDR_WIDTH-1):0]	AXI4-Lite Write Address Bus. The write address bus gives the address of the write transaction.
S_AXI_AWVALID	I	1	AXI4-Lite Write Address Channel Write Address Valid. This signal indicates that valid write address is available. <ul style="list-style-type: none"> <li>1 = Write address is valid.</li> <li>0 = Write address is not valid.</li> </ul>
S_AXI_AWREADY	O	1	AXI4-Lite Write Address Channel Write Address Ready. Indicates core is ready to accept the write address. <ul style="list-style-type: none"> <li>1 = Ready to accept address.</li> <li>0 = Not ready to accept address.</li> </ul>
<b>AXI Write Data Channel Signals(1)</b>			
S_AXI_WDATA	I	[(C_S_AXI_DATA_WIDTH-1):0]	AXI4-Lite Write Data Bus.
S_AXI_WSTRB	I	[C_S_AXI_DATA_WIDTH/8-1:0]	AXI4-Lite Write Strobes. This signal indicates which byte lanes to update in memory.
S_AXI_WVALID	I	1	AXI4-Lite Write Data Channel Write Data Valid. This signal indicates that valid write data and strobes are available. <ul style="list-style-type: none"> <li>1 = Write data/strobes are valid.</li> <li>0 = Write data/strobes are not valid.</li> </ul>
S_AXI_WREADY	O	1	AXI4-Lite Write Data Channel Write Data Ready. Indicates core is ready to accept the write data. <ul style="list-style-type: none"> <li>1 = Ready to accept data.</li> <li>0 = Not ready to accept data.</li> </ul>
<b>AXI Write Response Channel Signals(1)</b>			
S_AXI_BRESP(2)	O	[1:0]	AXI4-Lite Write Response Channel. Indicates results of the write transfer. <ul style="list-style-type: none"> <li>00b = OKAY - Normal access has been successful.</li> <li>01b = EXOKAY - Not supported.</li> <li>10b = SLVERR - Error.</li> <li>11b = DECERR - Not supported.</li> </ul>

Pin Name	Dir	Width	Description
S_AXI_BVALID	O	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid. <ul style="list-style-type: none"> <li>1 = Response is valid.</li> <li>0 = Response is not valid.</li> </ul>
S_AXI_BREADY	I	1	AXI4-Lite Write Response Channel Ready. Indicates Master is ready to receive response. <ul style="list-style-type: none"> <li>1 = Ready to receive response.</li> <li>0 = Not ready to receive response.</li> </ul>
<b>AXI Read Address Channel Signals(1)</b>			
S_AXI_ARADDR	I	[(C_S_AXI_ADDR_WIDTH-1):0]	AXI4-Lite Read Address Bus. The read address bus gives the address of a read transaction
S_AXI_ARVALID	I	1	AXI4-Lite Read Address Channel Read Address Valid. <ul style="list-style-type: none"> <li>1 = Read address is valid.</li> <li>0 = Read address is not valid.</li> </ul>
S_AXI_ARREADY	O	1	AXI4-Lite Read Address Channel Read Address Ready. Indicates core is ready to accept the read address. <ul style="list-style-type: none"> <li>1 = Ready to accept address.</li> <li>0 = Not ready to accept address.</li> </ul>
<b>AXI Read Data Channel Signals(1)</b>			
S_AXI_RDATA	O	[(C_S_AXI_DATA_WIDTH-1):0]	AXI4-Lite Read Data Bus.
S_AXI_RRESP(2)	O	[1:0]	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer. <ul style="list-style-type: none"> <li>00b = OKAY - Normal access has been successful.</li> <li>01b = EXOKAY - Not supported.</li> <li>10b = SLVERR - Error.</li> <li>11b = DECERR - Not supported.</li> </ul>
S_AXI_RVALID	O	1	AXI4-Lite Read Data Channel Read Data Valid. This signal indicates that the required read data is available and the read transfer can complete. <ul style="list-style-type: none"> <li>1 = Read data is valid.</li> <li>0 = Read data is not valid.</li> </ul>
S_AXI_RREADY	I	1	AXI4-Lite Read Data Channel Read Data Ready. Indicates master is ready to accept the read data. <ul style="list-style-type: none"> <li>1 = Ready to accept data.</li> <li>0 = Not ready to accept data.</li> </ul>

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

Descriptions of `gamma_reg2_status` and `gamma_reg3_addr_data` are detailed in "Updating the Gamma Tables Using the EDK pCore Interfaces."

### Double-Buffering

When using a Double-Buffered interface, all but the control and software reset registers are double-buffered in hardware to ensure no image tearing when gamma look-up table values are written by the processor during the active area of a frame. One buffer, the active buffer, is used actively by the core, while the inactive buffer can be updated by the processor. Double-buffering provides a more flexible and easier-to-use core because it decouples updating of the gamma tables from the video signal blanking period. This allows software to update the gamma tables at any time within the video frame, without degrading or corrupting the image quality. Using double-buffering may effectively double the number of block RAM primitives allocated by the core depending on the configuration of the core.

**Note:** When updating look-up tables using a double-buffered interface, it is highly recommended that the entire look-up table for all color channels be updated before the table is committed by asserting the Table Update bit (Bit 1 of `gamma_reg0_control`).

Due to the swapping of inactive and active look-up tables, performing only partial table updates may produce unexpected results.

### Single-Buffering

The option for single-buffered registers can cause image tearing or corruption if the tables are not fully updated in the blanking period. If the end-product can tolerate partially updated gamma tables, or the processor can ensure complete updates for the entire gamma table(s) during the vertical blanking period, a single-buffered interface can be used. This single-buffer configuration offers a savings in the number of block RAMs but trades off a tighter period that registers can be updated without detrimental video effects.

### Updating the Gamma Tables Using the EDK pCore Interfaces

The double- and single-buffered interfaces require that each write operation contain a valid address and data. Bits [31-16] of register `gamma_reg3_addr_data` are designated as the look-up table address, while bits [15-0] represent the number of words to be written into the gamma look-up table(s). The valid address range for the data depends on the input width, number of shared look-up tables, and whether interpolation is used, as shown in [Table 2-3](#).

Table 2-3: Valid Address Ranges for Gamma Correction Look-up Table Contents

Input Width	Look-up Tables*	Interpolation	Red/U/Cr Baseaddr, Range	Blue/V/Cb Baseaddr, Range	Green/Y Baseaddr, Range
8	3	0	0x0000, 0x00FF	0x0100, 0x01FF	0x0200, 0x02FF
8	2	0	0x0000, 0x00FF	N/A	0x0200, 0x02FF
8	1	0	0x0000, 0x00FF	N/A	N/A
10	3	0	0x0000, 0x03FF	0x0400, 0x07FF	0x0800, 0x0BFF
10	2	0	0x0000, 0x03FF	N/A	0x0800, 0x0BFF
10	1	0	0x0000, 0x03FF	N/A	N/A
12	3	0	0x0000, 0x0FFF	0x1000, 0x1FFF	0x2000, 0x2FFF
12	2	0	0x0000, 0x0FFF	N/A	0x2000, 0x2FFF

Table 2-3: Valid Address Ranges for Gamma Correction Look-up Table Contents (Cont'd)

Input Width	Look-up Tables*	Interpolation	Red/U/Cr Baseaddr, Range	Blue/V/Cb Baseaddr, Range	Green/Y Baseaddr, Range
12	1	0	0x0000, 0x0FFF	N/A	N/A
12	3	1	0x0000, 0x03FF	0x0400, 0x07FF	0x0800, 0x0BFF
12	2	1	0x0000, 0x03FF	N/A	0x0800, 0x0BFF
12	1	1	0x0000, 0x03FF	N/A	N/A

\* The number of lookup tables used is as follows:  
 3: when Independent look-up tables for each Color Channel is selected  
 2: when Identical look-up tables for Chrominance Channels Only is selected  
 1: when Identical look-up tables for all Color Channels is selected

## General Purpose Processor Interface

The General Purpose Processor Interface exposes the address and data buses, necessary to initialize look-up table contents, as ports. The Core Symbol for the General Purpose Processor Interface is shown in Figure 2-3. The Xilinx Streaming Video Interface is described in the previous section. The ports are described in Table 2-4.

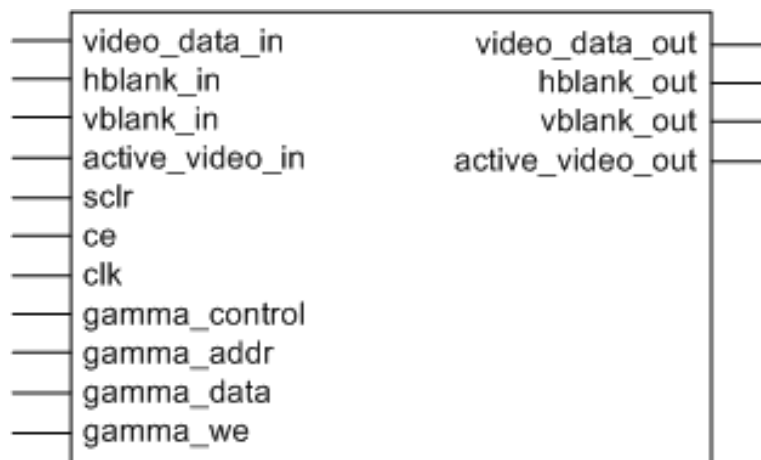


Figure 2-3: Core Symbol for the General Purpose Processor Interface

Table 2-4: Optional Ports for the General Purpose Processor Interface

Port Name	Port Width	Direction	Description
gamma_control	2	IN	Look-up table update, software enable
gamma_addr	14	IN	Look-up table address bus (2 additional MSBs are necessary only when no optimizations are used)
gamma_data	OWIDTH	IN	Look-up table initialization data corresponding to address provided by gamma_addr
gamma_we	1	IN	Look-up table write enable

- gamma\_control:** This port conveys the functionality of the EDK interface register `gamma_reg0_control`, defined in Table 2-2. The Software Enable bit can enable (1) or disable (0) core functionality. The Look-up Table Update bit, if asserted (1), will

trigger the update of the active look-up tables with the contents of the inactive look-up tables at the next rising edge of VBlank.

- **gamma\_addr:** Address bus for writing the gamma correction look-up table contents. This bus is always 14 bits wide, but the valid address range for accessing the internal look-up table contents varies depending on configuration. [Table 2-3](#) shows the valid address ranges for each configuration:
- **gamma\_data:** Data bus for writing internal look-up table contents.
- **gamma\_we:** Active '1' on `gamma_we` enables writing `gamma_data` into the look-up table address specified by `gamma_addr`.

# Customizing and Generating the Core

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

## Graphical User Interface (GUI)

The Gamma Correction core is easily configured to meet the user's specific needs through the CORE Generator GUI. This section provides a quick reference to parameters that can be configured at generation time. [Figure 3-1](#) shows the main Gamma Correction screen.

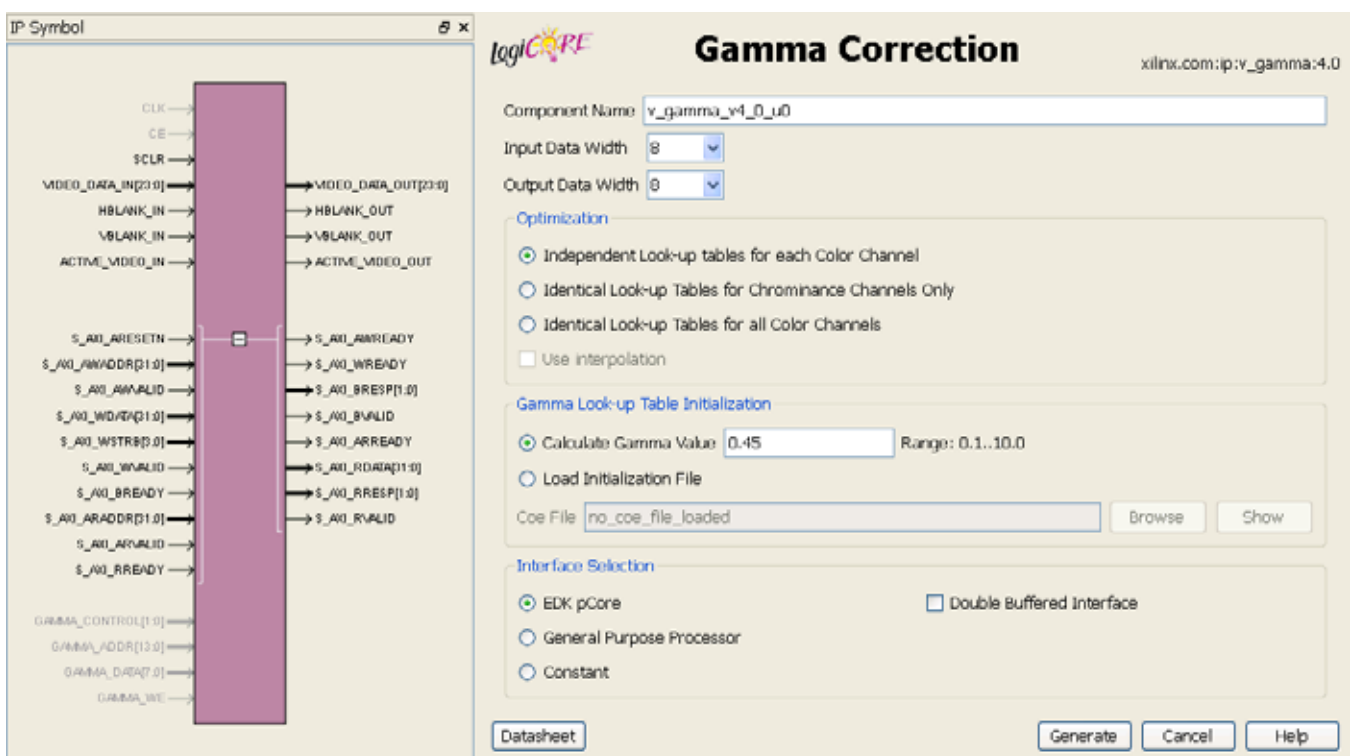


Figure 3-1: Gamma Correction Main Screen

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

- **Component Name:** The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and “\_”. **The name v\_gamma\_v4\_0 is not allowed.**

- **Input Data Width (IWIDTH):** Specifies the bit width of the input color channel for each component. Permitted values are 8, 10 and 12 bits.
- **Output Data Width (OWIDTH):** Specifies the bit width of the output color channel for each component. Permitted values are 8, 10 and 12 bits.
- **Optimization:** Specifies options to reduce memory usage.
  - **Independent Look-up Tables for each Color Channel:** When selected, each color channel uses a separate look-up table, permitting each channel to implement a distinct function. This option requires the most Block RAM resources (see [Optimization Options in Chapter 1](#)).
  - **Identical Look-up Tables for Chrominance Channels Only:** When selected, the chrominance channels (Cr, Cb) will share the same look-up table contents. (This also applies to the U and V channels for YUV or the Red and Blue channels of RGB). When two channels can use the same look-up table, the required number of write operations to modify the function stored in the look-up tables is reduced, and in some cases the number of Block RAM resources required is reduced (see [Optimization Options in Chapter 1](#)).
  - **Identical Look-up Tables for all Color Channels:** When selected, the red, green, and blue (or luminance and chrominance channels) all share the same look-up table contents. When all channels can use the same look-up tables, the required number of write operations to modify the function stored in the look-up tables is reduced, and in some cases the number of Block RAM resources required is reduced (see [Optimization Options in Chapter 1](#)).
  - **Use Interpolation:** Interpolation is used to reduce block RAM counts when using 12-bit input from 4k entries per look-up table to only 1k per look-up table (see [Optimization Options in Chapter 1](#)).
- **Gamma Look-Up Table Initialization**
  - **Calculate Gamma Value:** When selected, specifies the gamma value for initializing the look-up tables. Permitted values are floating-point values from 0.1 to 10.
  - **Load Initialization File:** When selected, the Load Initialization File feature allows a custom COE file to be loaded which specifies the contents of the gamma look-up tables. This permits the Gamma Correction core to be used to implement any function for a variety of tasks.
- **Interface Selection:** As described previously in this document, this option allows for the configuration of four different interfaces for the core.
  - **Double-Buffered Interface:** Double-buffering is used to eliminate tearing of the output images by writing to an inactive look-up table, then providing the ability to swap inactive and active look-up tables. This feature is only available for the EDK pCore and General Purpose Processor Interfaces. However, using this feature may increase the memory footprint of the core. See [Double-Buffering in Chapter 2](#) and [Optimization Options in Chapter 1](#).
  - **EDK pCore:** CORE Generator generates a pCore that can be easily inserted into an EDK project as a hardware peripheral. Gamma table values are reprogrammable via the AXI4-Lite bus interface. Note that the Gamma Correction pCore cannot be modified from within the EDK, and must be regenerated from the CORE Generator to modify the core's configuration.
  - **General Purpose Processor Interface:** CORE Generator software will generate a set of ports to be used to program the core. See [General Purpose Processor Interface in Chapter 2](#).

- **Constant Interface:** The gamma tables are constant, and therefore no programming is necessary.

## Parameter Values in the XCO File

Table 3-1 defines valid entries for the XCO parameters. Xilinx strongly suggests that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator software GUI to configure the core and perform range and parameter value checking. The XCO parameters are helpful in defining the interface to other Xilinx tools.

Table 3-1: XCO Parameters

CO Parameter	Default	Valid Values
component_name	v_gamma_v4_0_u0	ASCII text using characters: a..z, 0..9 and "_" starting with a letter. Note: "v_gamma_v4_0" is not allowed.
interface_selection	EDK_Pcore	EDK_Pcore, General_Purpose Processor, Constant
iwidth	8	8, 10, 12
owidth	8	8, 10, 12
coe_file	no_coe_file_loaded	
default_gamma	0.45	0.1 to 10.0
double_buffer	false	false, true
interpolate	false	false, true
load_init_file	0	0, 1
num_luts	3	1, 2, 3

## Output Generation

The output files generated from the Xilinx CORE Generator software for the core depend upon whether the interface selection is set to EDK pCore or General Purpose Processor/Constant. The output files are placed in the project directory.

### EDK pCore Files

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

Component\_Name>

- drivers
  - gamma\_v4\_00\_a
    - data
    - build

- example
  - src
  - pcores
    - axi\_gamma\_v4\_00\_a
    - data
    - hdl
- vhdl

### File Details

<project directory>

This is the top-level directory. It contains xco and other assorted files.

Name	Description
<component_name>.xco	Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_flist.txt	A text file listing all of the output files produced when the customized core was generated in the CORE Generator software.

<project directory>/<component\_name>/pcores/axi\_gamma\_v4\_00\_a/data

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

< project directory>/<component\_name>/pcores/axi\_gamma\_v4\_00\_a/hdl/vhdl

This directory contains the HDL files that implement the pCore.

< project directory>/<component\_name>/drivers/gamma\_v4\_00\_a/data

This directory contains files that SDK uses to define the operation of the pCore's software driver.

< project directory>/<component\_name>/drivers/gamma\_v4\_00\_a/example

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

< project directory>/<component\_name>/drivers/gamma\_v4\_00\_a/src

This directory contains the source code of the pCore's software driver.

Name	Description
gamma.c	Provides the API access to all features of the device driver.
gamma.h	Provides the API access to all features of the device driver.

## General Purpose Processor or Constant Interface Files

When the interface selection is set to General Purpose Processor, CORE Generator will output the core as a netlist that can be inserted into a processor interface wrapper or instantiated directly in an HDL design. The output is placed in the <project director>.

### File Details

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

Name	Description
<component_name>_readme.txt	Readme file for the core.
<component_name>.ngc	The netlist for the core.
<component_name>.veo	The HDL template for instantiating the core.
<component_name>.vho	
<component_name>.v	The structural simulation model for the core. It is used for functionally simulating the core.
<component_name>.vhd	
<component_name>.xco	Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.

## ***Constraining the Core***

---

### **Required Constraints**

There are no required constraints for this core.

### **Device, Package, and Speed Grade Selections**

This core has not been characterized for use in low power devices.

### **Clock Frequencies**

There are no specific clock frequency requirements for this core other than the Maximum Frequency discussed in [Performance in Chapter 1](#). This core has not been characterized for use in Low Power Devices.

### **Clock Management**

There is only one clock for this core. For pCore users, the AXI interconnect handles the cross clock domain crossing.

### **Clock Placement**

There are no specific clock placement requirements for this core.

### **Banking**

There are no specific banking rules for this core.

### **Transceiver Placement**

There are no transceivers used in this core.

### **I/O Standard and Placement**

There are no specific I/O standard or placement requirements.

## Designing with the Core

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

### Control Signals and Timing

The propagation delay of the Gamma Correction core is always four clock cycles. Deasserting CE suspends processing, which may be useful for data-throttling, to temporarily cease processing of a video stream to match the delay of other processing components.

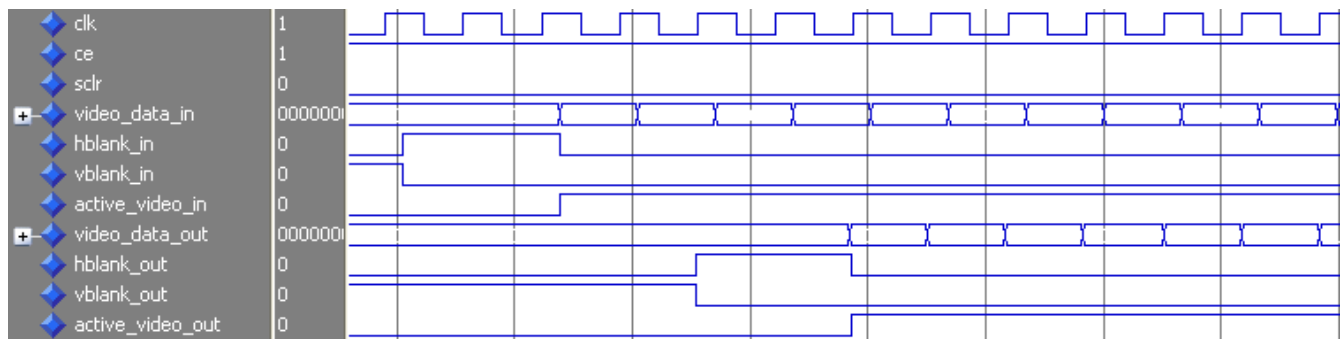


Figure 5-1: Timing Example

### Clocking

The Gamma Correction core has one clock ("clk") that is used to clock the entire core. This includes the AXI interface and the core logic.

### Resets

The Gamma Correction core has one reset ("sclr") that is used for the entire core. The reset is active high.

### Protocol Description

For the pCore version of the Gamma Correction core, the register interface is compliant with the AXI4-Lite interface.

# ***Verification, Compliance, and Interoperability***

---

## **Simulation**

A highly parameterizable test bench used to test the Gamma Correction core. Testing includes the following:

- Register accesses
- Testing table initialization with a gamma value
- Testing with initialization files
- Testing using 1, 2, or 3 look-up tables
- Testing interpolation for 12 bit input data
- Testing of the double buffering
- Testing of various data widths

## **Hardware Testing**

The Gamma Correction core has been tested in a variety of hardware platforms at Xilinx to represent a variety of parameterizations.

A test design was developed for the core that incorporated a MicroBlaze processor, AXI4-LITE Interface and various other peripherals.

# *Migrating*

---

## **Special Considerations when Migrating to AXI**

The Gamma Correction v4.0 interface changed from the PLB processor interface to the EDK pCore AXI4-Lite interface. As a result, all of the PLB-related connections have been replaced with an AXI4-Lite interface. For more information, see the AXI Reference Guide.

# *Debugging*

---

## **Evaluation Core Timeout**

The Gamma Correction hardware evaluation core times out after approximately 8 hours of operation. The output is driven to zero. This results in a black screen for RGB color systems and a dark-green screen for YUV color systems.

See [Solution Centers in Appendix F](#) for information helpful to the debugging progress.

## Application Software Development

---

### EDK pCore API Functions

This section describes the EDK pCore API functions.

**GAMMA\_Enable(uint32 BaseAddress);**

- This macro enables a Gamma instance.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from xparameters.h).

**GAMMA\_Disable(uint32 BaseAddress);**

- This macro disables a Gamma instance.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from xparameters.h).

**GAMMA\_RegUpdateEnable(uint32 BaseAddress);**

- When using a double-buffered interface, enabling the RegUpdateEnable causes the gamma correction look-up table to update on the next rising edge of vBlank\_in. This action causes the new values written to the inactive look-up table to become the active look-up table when vBlank\_in rising edge occurs. The user must manually disable the register update after a sufficient amount of time to prevent continuous updates.
- This function only works when the Gamma core is enabled.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from xparameters.h)

**GAMMA\_RegUpdateDisable(uint32 BaseAddress);**

- When using a double-buffered interface, disabling the Register Update prevents the gamma correction look-up table from updating. It is recommended that the Register Update be disabled while writing to the inactive look-up table in the gamma correction core, until the write operation is complete. While disabled, writes to the inactive look up table are stored, but do not effect the core's behavior.
- This function only works when the Gamma core is enabled.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from xparameters.h)

**GAMMA\_Reset(uint32 BaseAddress);**

- This macro resets a Gamma instance. This reset effects the core immediately, and may cause image tearing. This reset resets the Gamma's configuration registers, and holds the core's outputs in their reset state until GAMMA\_ClearReset() is called.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from xparameters.h)

**GAMMA\_ClearReset(uint32 BaseAddress);**

- This macro clears the Gamma's reset flag (which is set using `GAMMA_Reset()`), and returns it to normal operation. This ClearReset effects the core immediately, and may cause image tearing.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from `xparameters.h`).

**GAMMA\_AutoSyncReset(uint32 BaseAddress);**

- This macro resets a Gamma instance, but differs from `GAMMA_Reset()` in that it automatically synchronizes to the `vBlank_in` input of the core to prevent tearing. On the next rising-edge of `vBlank_in` following a call to `GAMMA_AutoSyncReset()`, all of the core's configuration registers and outputs are reset, then the reset flag is immediately released, allowing the core to immediately resume default operation.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from `xparameters.h`).

**GAMMA\_ReadReg(uint32 BaseAddress, uint32 RegOffset)**

- Read the given register.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from `xparameters.h`).
- RegOffset is the register offset of the register (defined in [Table 2-1](#)).
- This macro returns the 32-bit unsigned integer value of the register.

**GAMMA\_WriteReg(uint32 BaseAddress, uint32 RegOffset, uint32 Data)**

- Write the given register.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from `xparameters.h`).
- RegOffset is the register offset of the register (defined in [Table 2-1](#)).
- Data is the 32-bit value to write to the register.

**GAMMA\_GetIWIDTH(uint32 BaseAddress)**

- Read the Gamma's instantiation parameter IWIDTH from the Status register.
- Return the corresponding integer value.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from `xparameters.h`).

**GAMMA\_GetOWidth(uint32 BaseAddress)**

- Read the Gamma's instantiation parameter OWIDTH from the Status register.
- Return the corresponding integer value.
- BaseAddress is the Xilinx EDK base address of the Gamma core (from `xparameters.h`).

**GAMMA\_GetINTPOL(uint32 BaseAddress)**

- Read the Gamma's instantiation parameter INTPOL from the Status register.
- Return the corresponding integer value (1 if interpolation is used, 0 otherwise).
- BaseAddress is the Xilinx EDK base address of the Gamma core (from `xparameters.h`).

## C Model Reference

The bit accurate C model is accessed through a set of functions and data structures that are declared in the `v_gamma_v4_0_bitacc_cmodel.h` file.

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

```

struct xilinx_ip_v_gamma_v4_0_generics gamma_generics;
struct xilinx_ip_v_gamma_v4_0_inputs  gamma_inputs;
struct xilinx_ip_v_gamma_v4_0_outputs gamma_outputs;

```

The declaration of these structures is in the `v_gamma_v4_0_bitacc_cmodel.h` file.

[Table E-1](#) lists the generic parameters taken by the Gamma Correction v4.0 IP core bit accurate model, as well as the default values. For an actual instance of the core, these parameters can only be set in generation time through the CORE Generator™ GUI.

Table E-1: Model Generic Parameters and Default Values

Generic variable	Type	Default Value	Range	Description
IWIDTH	int	8	8,10,12	Input data width.
OWIDTH	int	8	8,10,12	Output width.
INTPOL	int	0	0, 1	1 indicates that the core uses interpolation. Enabled only when IWIDTH = 12.
IDENTICAL_TABLES	int	1	0,1	1 indicates that the core uses interpolation. Enabled only when IWIDTH = 8 or 10.
DEFAULT_GAMMA	double	0.45	0.1 – 10.0	Gamma value used to initialize the Gamma correction tables. Initialization applies to all core interface options.

Calling `xilinx_ip_v_gamma_v4_0_get_default_generics(&gamma_generics)` initializes the generics structure with the Gamma GUI defaults, listed in [Table E-1](#).

Table E-2: Core Generic Parameters and Default Values

Input Variable	Type	Default Value	Range	Description
video_in	video_struct	Null	N/A	Container to hold input image or video data. <sup>1</sup>
RTABLE	uint16[4096]	$\text{round}\left(255 \frac{k^{0.45}}{255^{0.45}}\right)$	0 to $2^{OWIDTH-1} - 1$	The R-, G-, and BTABLE variables hold the Gamma correction tables for the color channels.
GTABLE	uint16[4096]			
BTABLE	uint16[4096]			

<sup>1</sup> For the description of the input structure, see [Initializing the Gamma Correction Input Video Structure](#).

The structure `gamma_inputs` defines the values of run time parameters and the actual input image. The Red, Green, and Blue tables can be set dynamically through the pCore and General Purpose Processor interfaces. Consequently, these values are passed as inputs to the core, along with the actual test image, or video sequence (see [Table E-2](#)).

Calling `xilinx_ip_v_gamma_v4_0_get_default_inputs(&gamma_generics, &gamma_inputs)` initializes members of the input structure default values (see [Table E-2](#)).

**Note:** The `video_in` variable is not initialized because the initialization depends on the actual test image to be simulated. [Chapter 4, C Model Example Code](#) describes the initialization of the `video_in` structure.

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

```
int xilinx_ip_v_gamma_v4_0_bitacc_simulate(
    struct xilinx_ip_v_gamma_v4_0_generics* generics,
    struct xilinx_ip_v_gamma_v4_0_inputs* inputs,
    struct xilinx_ip_v_gamma_v4_0_outputs* outputs).
```

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

```
void xilinx_ip_v_gamma_v4_0_destroy(
    struct xilinx_ip_v_gamma_v4_0_inputs *input,
    struct xilinx_ip_v_gamma_v4_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.

## Gamma Correction Input and Output Video Structure

Input images or video streams can be provided to the Gamma Correction v4.0 reference model using the `video_struct` structure, defined in `video_utils.h`:

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

Table E-3: Member Variables of the Video Structure

Member Variable	Designation
frames	Number of video/image frames in the data structure.
rows	Number of rows per frame. Pertaining to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through all frames of the video stream. However different planes, such as y, u and v can have different dimensions.
cols	Number of columns per frame. Pertaining to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through all frames of the video stream. However different planes, such as y, u and v can have different dimensions.

Table E-3: Member Variables of the Video Structure (Cont'd)

bits_per_component	Number of bits per color channel/component. All image planes are assumed to have the same color/component representation. Maximum number of bits per component is 16.
mode	Contains information about the designation of data planes. Named constants to be assigned to mode are listed in <a href="#">Table E-4</a> .
data	Set of five pointers to three dimensional arrays containing data for image planes. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col].

Table E-4: Named Constants for Video Modes with Corresponding Planes and Representations <sup>(1)</sup>

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

1. The Gamma Correction core supports the FORMAT\_RGB and FORMAT\_C444 modes.

## Initializing the Gamma Correction 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 `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](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.

```
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:

```
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. If the output container structure is not pre-allocated, the utility functions create the appropriate container to hold results.

## Binary Image/Video Files

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

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

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

## Working with Video\_struct Containers

The video\_utils.h header file defines functions to simplify access to video data in video\_struct.

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

The video\_planes\_per\_mode function returns the number of component planes defined by the mode variable, as described in Table E-4. The video\_rows\_per\_plane and video\_cols\_per\_plane functions return the number of rows and columns in a given plane of the selected video structure. The following example demonstrates using these functions in conjunction to process all pixels within a video stream stored in the in\_video variable:

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

```
}  
}  
}
```

## Example Code

An example C file, `run_bitacc_cmodel.c`, is provided to demonstrate the steps required to run the model. After following the compilation instructions, run the example executable. The executable takes the path/name of the input file and the path/name of the output file as parameters. If invoked with insufficient parameters, this help message is issued:

```
Usage: run_bitacc_cmodel in_file out_file  
in_file      : path/name of the input  BMP file  
out_file     : path/name of the output BMP file
```

During successful execution, two files with a `.bin` extension are created. The first file corresponds to the input BMP image, with the same path and name as the input file, and a `.bin` extension. The other file similarly corresponds to the output file. These files contain the inputs and outputs of the Gamma Correction algorithm in full precision, as the BMP format does not support color resolutions beyond 8-bits per component. The structure of `.bin` files are described in [Binary Image/Video Files](#).

To ease modifying and debugging the provided top-level demonstrator using the built-in debugging environment of Visual Studio, the top-level command line parameters can be specified through the Project Property Pages using these steps:

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

## Compiling Gamma Correction C Model with Example Wrapper

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

To compile the example code, perform these steps:

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

```
setenv LD_LIBRARY_PATH <unzipped_c_model_dir>:${LD_LIBRARY_PATH}
```

2. Copy these files from the `/lin32` or `/lin64` directory to the root directory:

```
libstlport.so.5.1
```

```
libIp_v_gamma_v4_0_bitacc_cmodel.so
```

3. In the root directory, compile using the GNU C Compiler with this command:

```
gcc -m32 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o  
run_bitacc_cmodel -L. -lIp_v_gamma_v4_0_bitacc_cmodel -Wl,-rpath,.
```

```
gcc -m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o  
run_bitacc_cmodel -L. -lIp_v_gamma_v4_0_bitacc_cmodel -Wl,-rpath,.
```

## Windows (32- and 64-bit)

The precompiled library `v_gamma_v4_0_bitacc_cmodel.lib`, and top-level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. An example procedure is provided here using Microsoft Visual Studio.

1. In Visual Studio, create a new, empty Console Application project.
2. As existing items, add:
  - a. `libIp_v_gamma_v4_0_bitacc_cmodel.lib` to the Resource Files folder of the project
  - b. `run_bitacc_cmodel.c` to the Source Files folder of the project
  - c. `v_gamma_v4_0_bitacc_cmodel.h` to the Header Files folder of the project
3. After the project is created and populated, it must be compiled and linked (built) to create a executable. To perform the build step, select "Build Solution" from the Build menu. An executable matching the project name has been created either in the Debug or Release subdirectories under the project location based on whether "Debug" or "Release" has been selected in the "Configuration Manager" under the Build menu.

# ***Additional Resources***

---

## **Xilinx Resources**

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

<http://www.xilinx.com/support>.

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

[http://www.xilinx.com/support/documentation/sw\\_manuals/glossary.pdf](http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf).

## **Solution Centers**

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

## **References**

These documents provide supplemental material useful with this user guide:

1. [AXI Reference Guide](#).

## **Technical Support**

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

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

- New Features
- Resolved Issues
- Known Issues

## Ordering Information

The Gamma Correction core is provided under the [Xilinx Core License Agreement](#) and can be generated using the Xilinx® CORE Generator™ system. The CORE Generator system is shipped with Xilinx ISE® Design Suite software.

A simulation evaluation license for the core is shipped with the CORE Generator system. To access the full functionality of the core, including FPGA bitstream generation, a full license must be obtained from Xilinx. For more information, visit the [product page](#) for this core.

Contact your local Xilinx [sales representative](#) for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx [IP Center](#).

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release.

## Notice of Disclaimer

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 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.