# LogiCORE IP Video On-Screen Display v3.0

## *Product Guide*

**PG010 October 19, 2011**

XILINX®

# *Table of Contents*


## Chapter 1: Overview

**Standards Compliance** . . . . . 6
**Feature Summary** . . . . . 6
**Applications** . . . . . 7
**Unsupported Features** . . . . . 7
**Licensing** . . . . . 7
**Performance** . . . . . 8
**Resource Utilization** . . . . . 10

## Chapter 2: Core Interfaces and Register Space

**Port Descriptions** . . . . . 38
**I/O Interface and Timing** . . . . . 48
**Register Space** . . . . . 54

## Chapter 3: Customizing and Generating the Core

**GUI** . . . . . 64
**Parameter Values in the XCO File** . . . . . 68
**Parameter Modification in CORE Generator** . . . . . 69
**Output Generation** . . . . . 70

## Chapter 4: Designing with the Core

**General Design Guidelines** . . . . . 73
**Algorithm** . . . . . 78
**Clocking** . . . . . 79
**Resets** . . . . . 79
**Protocol Description** . . . . . 80

## Chapter 5: Constraining the Core

**Required Constraints** . . . . . 81
**Device, Package, and Speed Grade Selections** . . . . . 81
**Clock Frequencies** . . . . . 81
**Clock Management** . . . . . 81
**Clock Placement** . . . . . 81
**Banking** . . . . . 81
**Transceiver Placement** . . . . . 81
**I/O Standard and Placement** . . . . . 82

## Chapter 6: Detailed Example Design

Multi-FIFO to Stream (XSVI) Mode . . . 83
Multi-FIFO to FIFO Mode . . . 84
Directory and File Contents . . . 85
Demonstration Test Bench . . . 86
Simulation . . . 86
Messages and Warnings . . . 86

## Appendix A: Verification, Compliance, and Interoperability

Simulation . . . 87
Hardware Testing . . . 87

## Appendix B: Migrating

Migrating to the EDK pCore AXI4-Lite Interface . . . 89
Migrating to the AXI4-Stream Interface . . . 89
Parameter Changes in the XCO File . . . 89
Port Changes . . . 89
Functionality Changes . . . 89

## Appendix C: Debugging

## Appendix D: Application Software Development

Programming the Graphics Controller(s) . . . 92
EDK pCore Programmers Guide . . . 109
pCore API Functions . . . 110

## Appendix E: C Model Reference

Unpacking and Model Contents . . . 113
Installation . . . 115
Software Requirements . . . 115
Interface . . . 115
Example Code . . . 125

## Appendix F: Additional Resources

Xilinx Resources . . . 143
References . . . 143
Technical Support . . . 143
Ordering Information . . . 143
Revision History . . . 144
Notice of Disclaimer . . . 144

# LogiCORE IP Video On-Screen Display v3.0

## Introduction

The Xilinx LogiCORE™ IP Video On-Screen Display (OSD) provides a flexible video processing block for alpha blending and compositing as well as simple text and graphics generation. Support for up to eight layers using a combination of external video inputs (from frame buffer or streaming video cores via AXI4-Stream interfaces) and internal graphics controllers (including text generators) is provided. The core is programmable through a comprehensive register interface to set and control screen size, background color, layer position, and more using logic or a microprocessor. A comprehensive set of interrupt status bits is provided for processor monitoring.

## Features

- Supports Advanced eXtensible Interface (AXI4) Lite processor interface (EDK pCore) and AXI4-Stream data interfaces.
- Selectable processor interface: AXI4-Lite (EDK pCore) or General Purpose Processor (GPP)
- Supports video frame sizes up to 4096x4096 pixels
- Supports video frame rates up to 60fps (includes all HD rates such as 720p60 and 1080P60)
- Supports 2 or 3 color component channels
- Supports 8, 10 or 12 bits per color component channel (equivalent supported bits per pixel: 16, 20, 24, 30 or 36 bits)
- Supports alpha-blending 8 video/graphics layers
- Provides programmable background color
- Provides programmable layer position, size and z-plane order
- Generates filled and outlined transparent boxes
- Generates text with 1-bit or 2-bit per pixel color depth
- Provides configurable internal text string memory
- Provides configurable internal font memory for 8x8 or 16x16 pixel fixed distance fonts
- Provides scaling text by 1x, 2x, 4x or 8x
- Supports graphics color palette of 16 or 256 colors

| LogiCORE IP Facts Table | | | | | |
|---|---|---|---|---|---|
| **Core Specifics** | | | | | |
| Supported Device Family(1) | Virtex®-7, Kintex™-7, Virtex-6, Spartan®-6, | | | | |
| Supported User Interfaces | General Purpose Processor (GPP), EDK pCore AXI4-Lite | | | | |
| | **Resources** | | | | **Frequency** |
| Configuration | **LUTs** | **FFs** | **DSP Slices** | **Block RAMs** | **Max. Freq.** |
| Virtex-7 | See Table 1-3, page 11 | | | | 225 MHz |
| Kintex-7 | See Table 1-4, page 17 | | | | 150 MHz |
| Virtex-6 | See Table 1-5, page 24 | | | | 225 MHz |
| Spartan-6 | See Table 1-6, page 30 | | | | 150 MHz |
| **Provided with Core** | | | | | |
| Design Files | Netlist, EDK pCore files, C Driver | | | | |
| Example Design | Not Provided | | | | |
| Test Bench | VHDL(2) | | | | |
| Constraints File | Not Provided | | | | |
| Simulation Model | VHDL or Verilog Structural Model<br>C Model(2) | | | | |
| **Tested Design Tools** | | | | | |
| Design Entry Tools | ISE® Design Suite 13.3<br>Xilinx Platform Studio (XPS) 13.3 | | | | |
| Simulation(3) | Mentor Graphics® ModelSim, Xilinx ISim 13.3 | | | | |
| Synthesis Tools | Xilinx Synthesis Technology (XST) 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 Video OSD product page.
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

Chapter 1

# Overview

The Xilinx LogiCORE™ IP Video On-Screen Display (OSD) produces output video from multiple external video sources and multiple internal graphics controllers. Each graphics controller generates simple text and graphics overlays. Each video and graphics source is assigned an image layer. Up to eight image layers can be dynamically positioned, resized, brought forward or backward, and combined using alpha-blending.

Alpha-blending is the convex combination of two image layers allowing for transparency. Each layer in the OSD has a definite Z-plane order; or conceptually, each layer resides closer or farther from the observer having a different depth. Thus, the image and the image directly "over" it are blended. The order and amount of blending is programmable in real-time.

An example Xilinx Video On-Screen Display Output is shown in Figure 1-1.


*Figure 1-1:* **Example of OSD Output**

Figure 1-1 shows an example OSD output with multiple video and graphics layers. The three video layers (Video 1, 2 and 3) can be still images or live video, and are combined with transparency to the programmable background color. Simple boxes and text are generated with one or multiple internal graphics controllers (shown with yellow text and menu buttons) and are blended with the other layers. Another video layer (the Xilinx logo), can be generated from on-chip or external memory, showing that the OSD output can be easily extended with external logic, a microprocessor, or memory storage.

## Standards Compliance

The On-Screen Display core is compliant with the AXI4-Lite and AXI4-Stream standards as defined in UG761, *Xilinx AXI Reference Guide*.

## Feature Summary

The Video On-Screen Display core supports the Advanced eXtensible Interface Lite (AXI4-Lite) and the General Purpose Processor (GPP) interfaces. The AXI4-Lite interface can be easily incorporated into an EDK project. The GPP interface exposes the core registers to the user. The user can wrap the exposed registers in an interface that is compliant with user system. These configurable interfaces allow the OSD to be integrated easily with AXI4 processor-based systems, non-AXI4-compliant processor systems with little logic, and systems without a processor.

In addition, the OSD supports the AXI4-Stream and XSVI input on the input interfaces. Each input layer can be configured to accept AXI4-Stream or XSVI input (only one XSVI input is available, but this can be routed to any layer or to multiple layers). The configurable input interfaces allow easy integration with other Xilinx Video IP cores including the AXI VDMA, Video Scaler, Color Space Converters, Chroma Resampler and Video Timing Controller. Other AXI4-Stream Video IP is also supported.

The Video On-Screen Display core is capable of operating at frequencies beyond those for 1080p60 or 1080p50 with 2 or 3 color components channels at 8, 10 or 12 bits per color component channel (equivalent supported bits per pixel: 16, 20, 24, 30 or 36 bits). This allows frame sizes up to 4096 x 4096 pixels to be displayed. The OSD also accepts up to eight input sources and performs alpha blending. The user can configure multiple input video sources from AXI4-Stream, XSVI input or external memory through the AXI VDMA. Each video source layer can be displayed at different cropped sizes, positions, and transparency to a programmable background color and other layers. In addition, each source layer can be displayed on top of or below other layers with a few register writes. Each layer can use pixel-level alpha values to enable non-rectangular masks and non-rectangular graphics overlays.

When using the Video On-Screen Display core, the eight video layers are not limited to external sources. The OSD also allows instantiating a set of internal graphics controllers. Each layer can be driven by a graphics controller, and each graphics controller can be configured independently. The graphics controllers contain box and text generators that can be reconfigured at runtime to move or resize text and boxes. Boxes can be filled or outlined and the outline width is configurable. Text is generated from an internal font that the user can load or reload at run time. Text can also be scaled up to eight times of the internal font with two or four colors for each string on the screen. The graphics controllers can be configured for 16 or 256 colors, and each color has an independent transparency alpha value. The runtime configurability of the graphics controller allows the user to generate dynamic animated displays that blend seamlessly with multiple video sources.

# Applications

Applications range from broadcast and consumer to automotive, medical and industrial imaging and can include:

- Video Surveillance
- Machine Vision
- Video Conferencing
- Set-top box displays

# Unsupported Features

The Video On-Screen Display core does not natively convert input layer data color spaces. The OSD expects all input layers to be the same format as the output. However, video data with different color spaces can be used with the OSD with the addition of the Xilinx RGB-to-YCrCb, Xilinx YCrCb-to-RGB and Xilinx Chroma Resampler cores.

# Licensing

The Xilinx Video On-Screen Display LogiCORE system provides three licensing options. After installing the required Xilinx ISE software, 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 the core functionality with your own design and demonstrates the various interfaces on the core in simulation. (Functional simulation is supported by a dynamically-generated HDL structural model.)

## 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 back-annotated gate-level simulation of the core using the 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. This core is configured to time out after 8 hours of operation.

## Full

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

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

### Obtaining Your License

This section contains information about obtaining a simulation, full system hardware, and full license keys.

#### Simulation License

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 License

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.

#### Obtaining a Full License

To obtain a Full license key, purchase a license for the core. After doing so, click the "Access 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

This section contains data about the typical performance of the Video On-Screen Display core.

### 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 FPGA device, using a different version of Xilinx tools, and other factors.

- Virtex®-7 FPGA: 225 MHz
- Kintex™-7 FPGA: 150 MHz
- Virtex-6 FPGA: 225 MHz
- Spartan®-6 FPGA: 150 MHz

### Latency

The Video On-Screen Display core can be configured for XSVI or AXI4-Stream input interfaces. The output can also be an XSVI or AXI4-Stream output. The latency from XSVI input to XSVI output data is 16 + 4*C_NUM_LAYERS. The latency to and from

AXI4-Stream interfaces is a minimum of 16 + 4*C_NUM_LAYERS, but `tready` and `tvalid` will increase the overall latency of the core. The number of layers affects the latency. Each layer (configured by C_NUM_LAYERS) adds approximately four cycles.

## Throughput

The Video On-Screen Display core throughput is mostly limited by the clock frequency and frame size (4096 x 4096 pixels). The other limiting factor is that the OSD also requires one extra line of initialization time each frame. This time is usually absorbed by the vertical blanking period for XSVI output, but will marginally lower the overall throughput for AXI4-Stream output (the AXI4-Stream throughput is usually higher as compared to XSVI output because the AXI4-Stream does not require horizontal/vertical blanking time). Vertical and horizontal blanking will limit the throughput with XSVI output.

The typical maximum output throughput (AXI4-Stream output) is calculated by Equation 1-1.

$$\frac{\text{cycles per second} \cdot \text{lines per frame} \cdot \text{channels per pixel} \cdot \text{bits per channel}}{\text{cycles per frame}} \quad \textit{Equation 1-1}$$

For AXI4-Stream output, this reduces to Equation 1-2:

$$\frac{\text{cycles per second} \cdot 4096 \cdot \text{channels per pixel} \cdot \text{bits per channel}}{4097} \quad \textit{Equation 1-2}$$

For XSVI output, the total number of cycles per frame (including horizontal and vertical blanking periods) must be known. The channels per pixel * bits per channel is the bits per pixel. Table 1-1 shows the maximum achievable output throughput for the different target frequencies for AXI4-Stream interface.

*Table 1-1:* **AXI4-Stream Throughput**

| Channels | Alpha Channel | Channel Data Width | Bits per Pixel | Max Throughput $F_{MAX}$ = 150 MHz (Mbits/s) | Max Throughput $F_{MAX}$ = 225 MHz (Mbits/s) |
|---|---|---|---|---|---|
| 2 | 0 | 8 | 16 | 2399414206 | 3599121308 |
| 2 | 0 | 10 | 20 | 2999267757 | 4498901635 |
| 2 | 0 | 12 | 24 | 3599121308 | 5398681962 |
| 3 | 0 | 8 | 24 | 3599121308 | 5398681962 |
| 3 | 0 | 10 | 30 | 4498901635 | 6748352453 |
| 3 | 0 | 12 | 36 | 5398681962 | 8098022944 |
| 2 | 1 | 8 | 24 | 3599121308 | 5398681962 |
| 2 | 1 | 10 | 30 | 4498901635 | 6748352453 |
| 2 | 1 | 12 | 36 | 5398681962 | 8098022944 |
| 3 | 1 | 8 | 32 | 4798828411 | 7198242617 |
| 3 | 1 | 10 | 40 | 5998535514 | 8997803271 |
| 3 | 1 | 12 | 48 | 7198242617 | 10797363925 |

In addition, the Video On-Screen Display core pads all input and output AXI4-Stream interfaces to the nearest power of 2 (16, 32 or 64 bits). Table 1-2 shows the maximum achievable output throughput with the padding bits included.

*Table 1-2:* **AXI4-Stream Throughput with Padding Bits**

| Channels | Alpha Channel | Channel Data Width | Bits per Pixel | Max Throughput $F_{MAX}$ = 150 MHz (Mbits/s) | Max Throughput $F_{MAX}$ = 225 MHz (Mbits/s) |
|---|---|---|---|---|---|
| 2 | 0 | 8 | 16 | 2399414206 | 3599121308 |
| 2 | 0 | 10 | 32 | 4798828411 | 7198242617 |
| 2 | 0 | 12 | 32 | 4798828411 | 7198242617 |
| 3 | 0 | 8 | 32 | 4798828411 | 7198242617 |
| 3 | 0 | 10 | 32 | 4798828411 | 7198242617 |
| 3 | 0 | 12 | 64 | 9597656822 | 14396485233 |
| 2 | 1 | 8 | 32 | 4798828411 | 7198242617 |
| 2 | 1 | 10 | 32 | 4798828411 | 7198242617 |
| 2 | 1 | 12 | 64 | 9597656822 | 14396485233 |
| 3 | 1 | 8 | 32 | 4798828411 | 7198242617 |
| 3 | 1 | 10 | 64 | 9597656822 | 14396485233 |
| 3 | 1 | 12 | 64 | 9597656822 | 14396485233 |

This can be compared to the user required throughput for any given video size by performing the calculation shown in Equation 1-3.

$$\text{frames per second} \cdot \text{lines per frame} \cdot \text{pixels per line} \cdot \text{channels per pixel} \cdot \text{bits per channel} \qquad \textit{Equation 1-3}$$

# Resource Utilization

Resources required for devices are estimated in Table 1-3 through Table 1-6 and use the same configuration for estimating resources for Virtex-7, Kintex-7, Virtex-6, and Spartan-6 devices.

Resource usage values were generated using the Xilinx CORE Generator in ISE® 13.3 tools. They are derived from post-MAP reports, but may change due to optimization settings or post-PAR optimization. All numbers are generated using the General Purpose Processor Interface of the Xilinx Video On-Screen Display.

All resource estimate configurations containing Graphics Controller layers have the Graphics Controller parameters set to the following:

- Instructions = 48
- Number of Colors = 16
- Number of Characters = 96
- Character Width = 8
- Character Height = 8
- ASCII Offset = 32

- Character Bits per Pixel = 1
- Number of Strings = 8
- Maximum String Length = 32

Different Graphics Controller parameter settings affect block RAM utilization. The following equation yields the upper bound of the block RAM utilization for Virtex-5 and Virtex-6 devices. The actual utilization may be lower due to block RAM data packing.

*Number of Block RAMs <=*

*(Maximum Screen Width) * LOG2(Number of Colors) /8192*

*+ Instructions / 128*

*+ (Number of Characters) * (Character Width) * (Character Height) * (Character Bits per Pixel) / 8192*

*+ (Number of Strings) * (Maximum String Length) / 1024*

The following equation yields the upper bound of the block RAM utilization for Spartan-3A DSP and Spartan-6 devices. The actual utilization may be lower due to block RAM data packing.

*Number of Block RAMs <=*

*(Maximum Screen Width) * LOG-2(Number of Colors) /4096*

*+ Instructions / 128*

*+ (Number of Characters) * (Character Width) * (Character Height) * (Character Bits per Pixel) / 8192*

*+ (Number of Strings) * (Maximum String Length) / 1024*

The EDK pCore interface adds an estimated additional 1130 FFs and 550 LUTs for configurations registers and the AXI4-Lite interface. The Maximum Screen Width parameter does not affect the AXI4-Stream or XSVI input layer resources.

Table 1-3 shows the resource estimates for Virtex-7 devices, and Table 1-4 shows the resource estimates for Kintex-7 devices. Table 1-5 shows the resource estimates for Virtex-6 devices, and Table 1-6 shows the resource estimates for Spartan-6 devices.

*Table 1-3:* **Virtex-7 Resource Estimates**

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 8 | 2 | 1 | 320 | 2 | 2 | 1,215 | 1,350 |
| Graphics Controller | 8 | 2 | 1 | 4096 | 2 | 3 | 1,214 | 1,362 |
| Graphics Controller | 8 | 2 | 2 | 320 | 4 | 4 | 2,227 | 2,442 |
| Graphics Controller | 8 | 2 | 2 | 4096 | 4 | 6 | 2,228 | 2,466 |
| Graphics Controller | 8 | 2 | 3 | 320 | 6 | 6 | 3,375 | 3,732 |
| Graphics Controller | 8 | 2 | 3 | 4096 | 6 | 9 | 3,369 | 3,769 |
| Graphics Controller | 8 | 2 | 4 | 320 | 8 | 8 | 4,441 | 4,942 |
| Graphics Controller | 8 | 2 | 4 | 4096 | 8 | 12 | 4,468 | 4,991 |
| Graphics Controller | 8 | 2 | 5 | 320 | 10 | 10 | 5,752 | 6,868 |

*Table 1-3:* **Virtex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 8 | 2 | 5 | 4096 | 10 | 15 | 5,841 | 6,932 |
| Graphics Controller | 8 | 2 | 6 | 320 | 12 | 12 | 6,980 | 8,294 |
| Graphics Controller | 8 | 2 | 6 | 4096 | 12 | 18 | 7,044 | 8,371 |
| Graphics Controller | 8 | 2 | 7 | 320 | 14 | 14 | 8,283 | 9,907 |
| Graphics Controller | 8 | 2 | 7 | 4096 | 14 | 21 | 8,260 | 9,997 |
| Graphics Controller | 8 | 2 | 8 | 320 | 16 | 16 | 9,581 | 11,417 |
| Graphics Controller | 8 | 2 | 8 | 4096 | 16 | 24 | 9,525 | 11,520 |
| Graphics Controller | 8 | 3 | 1 | 320 | 3 | 2 | 1,273 | 1,458 |
| Graphics Controller | 8 | 3 | 1 | 4096 | 3 | 3 | 1,241 | 1,470 |
| Graphics Controller | 8 | 3 | 2 | 320 | 6 | 4 | 2,375 | 2,647 |
| Graphics Controller | 8 | 3 | 2 | 4096 | 6 | 6 | 2,371 | 2,672 |
| Graphics Controller | 8 | 3 | 3 | 320 | 9 | 6 | 3,592 | 4,084 |
| Graphics Controller | 8 | 3 | 3 | 4096 | 9 | 9 | 3,529 | 4,122 |
| Graphics Controller | 8 | 3 | 4 | 320 | 12 | 8 | 4,722 | 5,406 |
| Graphics Controller | 8 | 3 | 4 | 4096 | 12 | 12 | 4,788 | 5,456 |
| Graphics Controller | 8 | 3 | 5 | 320 | 15 | 10 | 6,253 | 7,614 |
| Graphics Controller | 8 | 3 | 5 | 4096 | 15 | 15 | 6,229 | 7,679 |
| Graphics Controller | 8 | 3 | 6 | 320 | 18 | 12 | 7,504 | 9,194 |
| Graphics Controller | 8 | 3 | 6 | 4096 | 18 | 18 | 7,538 | 9,272 |
| Graphics Controller | 8 | 3 | 7 | 320 | 21 | 14 | 8,832 | 11,011 |
| Graphics Controller | 8 | 3 | 7 | 4096 | 21 | 21 | 8,881 | 11,100 |
| Graphics Controller | 8 | 3 | 8 | 320 | 24 | 16 | 10,160 | 12,684 |
| Graphics Controller | 8 | 3 | 8 | 4096 | 24 | 24 | 10,202 | 12,787 |
| Graphics Controller | 10 | 2 | 1 | 320 | 2 | 2 | 1,237 | 1,457 |
| Graphics Controller | 10 | 2 | 1 | 4096 | 2 | 3 | 1,263 | 1,469 |
| Graphics Controller | 10 | 2 | 2 | 320 | 4 | 4 | 2,316 | 2,638 |
| Graphics Controller | 10 | 2 | 2 | 4096 | 4 | 6 | 2,342 | 2,663 |
| Graphics Controller | 10 | 2 | 3 | 320 | 6 | 6 | 3,459 | 4,061 |
| Graphics Controller | 10 | 2 | 3 | 4096 | 6 | 9 | 3,503 | 4,099 |
| Graphics Controller | 10 | 2 | 4 | 320 | 8 | 8 | 4,583 | 5,384 |
| Graphics Controller | 10 | 2 | 4 | 4096 | 8 | 12 | 4,640 | 5,434 |
| Graphics Controller | 10 | 2 | 5 | 320 | 10 | 10 | 6,103 | 7,548 |

*Table 1-3:* **Virtex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 10 | 2 | 5 | 4096 | 10 | 15 | 6,121 | 7,612 |
| Graphics Controller | 10 | 2 | 6 | 320 | 12 | 12 | 7,411 | 9,126 |
| Graphics Controller | 10 | 2 | 6 | 4096 | 12 | 18 | 7,444 | 9,202 |
| Graphics Controller | 10 | 2 | 7 | 320 | 14 | 14 | 8,581 | 10,932 |
| Graphics Controller | 10 | 2 | 7 | 4096 | 14 | 21 | 8,722 | 11,022 |
| Graphics Controller | 10 | 2 | 8 | 320 | 16 | 16 | 10,093 | 12,612 |
| Graphics Controller | 10 | 2 | 8 | 4096 | 16 | 24 | 10,134 | 12,715 |
| Graphics Controller | 10 | 3 | 1 | 320 | 3 | 2 | 1,318 | 1,587 |
| Graphics Controller | 10 | 3 | 1 | 4096 | 3 | 3 | 1,296 | 1,600 |
| Graphics Controller | 10 | 3 | 2 | 320 | 6 | 4 | 2,432 | 2,884 |
| Graphics Controller | 10 | 3 | 2 | 4096 | 6 | 6 | 2,465 | 2,909 |
| Graphics Controller | 10 | 3 | 3 | 320 | 9 | 6 | 3,700 | 4,483 |
| Graphics Controller | 10 | 3 | 3 | 4096 | 9 | 9 | 3,689 | 4,520 |
| Graphics Controller | 10 | 3 | 4 | 320 | 12 | 8 | 4,943 | 5,941 |
| Graphics Controller | 10 | 3 | 4 | 4096 | 12 | 12 | 4,954 | 5,991 |
| Graphics Controller | 10 | 3 | 5 | 320 | 15 | 10 | 6,567 | 8,452 |
| Graphics Controller | 10 | 3 | 5 | 4096 | 15 | 15 | 6,565 | 8,516 |
| Graphics Controller | 10 | 3 | 6 | 320 | 18 | 12 | 8,066 | 10,216 |
| Graphics Controller | 10 | 3 | 6 | 4096 | 18 | 18 | 7,974 | 10,292 |
| Graphics Controller | 10 | 3 | 7 | 320 | 21 | 14 | 9,361 | 12,272 |
| Graphics Controller | 10 | 3 | 7 | 4096 | 21 | 21 | 9,464 | 12,362 |
| Graphics Controller | 10 | 3 | 8 | 320 | 24 | 16 | 10,768 | 14,148 |
| Graphics Controller | 10 | 3 | 8 | 4096 | 24 | 24 | 10,884 | 14,251 |
| Graphics Controller | 12 | 2 | 1 | 320 | 2 | 2 | 1,290 | 1,533 |
| Graphics Controller | 12 | 2 | 1 | 4096 | 2 | 3 | 1,290 | 1,546 |
| Graphics Controller | 12 | 2 | 2 | 320 | 4 | 4 | 2,427 | 2,775 |
| Graphics Controller | 12 | 2 | 2 | 4096 | 4 | 6 | 2,441 | 2,800 |
| Graphics Controller | 12 | 2 | 3 | 320 | 6 | 6 | 3,701 | 4,299 |
| Graphics Controller | 12 | 2 | 3 | 4096 | 6 | 9 | 3,666 | 4,337 |
| Graphics Controller | 12 | 2 | 4 | 320 | 8 | 8 | 4,854 | 5,705 |
| Graphics Controller | 12 | 2 | 4 | 4096 | 8 | 12 | 4,952 | 5,755 |
| Graphics Controller | 12 | 2 | 5 | 320 | 10 | 10 | 6,332 | 8,089 |

*Table 1-3:* **Virtex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 12 | 2 | 5 | 4096 | 10 | 15 | 6,426 | 8,153 |
| Graphics Controller | 12 | 2 | 6 | 320 | 12 | 12 | 7,845 | 9,791 |
| Graphics Controller | 12 | 2 | 6 | 4096 | 12 | 18 | 7,813 | 9,867 |
| Graphics Controller | 12 | 2 | 7 | 320 | 14 | 14 | 9,173 | 11,762 |
| Graphics Controller | 12 | 2 | 7 | 4096 | 14 | 21 | 9,184 | 11,852 |
| Graphics Controller | 12 | 2 | 8 | 320 | 16 | 16 | 10,588 | 13,584 |
| Graphics Controller | 12 | 2 | 8 | 4096 | 16 | 24 | 10,736 | 13,687 |
| Graphics Controller | 12 | 3 | 1 | 320 | 3 | 2 | 1,363 | 1,685 |
| Graphics Controller | 12 | 3 | 1 | 4096 | 3 | 3 | 1,388 | 1,698 |
| Graphics Controller | 12 | 3 | 2 | 320 | 6 | 4 | 2,548 | 3,059 |
| Graphics Controller | 12 | 3 | 2 | 4096 | 6 | 6 | 2,565 | 3,084 |
| Graphics Controller | 12 | 3 | 3 | 320 | 9 | 6 | 3,882 | 4,791 |
| Graphics Controller | 12 | 3 | 3 | 4096 | 9 | 9 | 3,854 | 4,828 |
| Graphics Controller | 12 | 3 | 4 | 320 | 12 | 8 | 5,126 | 6,635 |
| Graphics Controller | 12 | 3 | 4 | 4096 | 12 | 12 | 5,200 | 6,687 |
| Graphics Controller | 12 | 3 | 5 | 320 | 15 | 10 | 6,809 | 9,147 |
| Graphics Controller | 12 | 3 | 5 | 4096 | 15 | 15 | 6,833 | 9,211 |
| Graphics Controller | 12 | 3 | 6 | 320 | 18 | 12 | 8,438 | 11,069 |
| Graphics Controller | 12 | 3 | 6 | 4096 | 18 | 18 | 8,471 | 11,145 |
| Graphics Controller | 12 | 3 | 7 | 320 | 21 | 14 | 10,039 | 13,335 |
| Graphics Controller | 12 | 3 | 7 | 4096 | 21 | 21 | 9,930 | 13,425 |
| Graphics Controller | 12 | 3 | 8 | 320 | 24 | 16 | 11,639 | 15,381 |
| Graphics Controller | 12 | 3 | 8 | 4096 | 24 | 24 | 11,675 | 15,484 |
| AXI4-Stream | 8 | 2 | 1 | - | 2 | 0 | 411 | 625 |
| AXI4-Stream | 8 | 2 | 2 | - | 4 | 0 | 628 | 1,033 |
| AXI4-Stream | 8 | 2 | 3 | - | 6 | 0 | 991 | 1,629 |
| AXI4-Stream | 8 | 2 | 4 | - | 8 | 0 | 1,270 | 2,148 |
| AXI4-Stream | 8 | 2 | 5 | - | 10 | 0 | 1,821 | 3,141 |
| AXI4-Stream | 8 | 2 | 6 | - | 12 | 0 | 2,305 | 3,828 |
| AXI4-Stream | 8 | 2 | 7 | - | 14 | 0 | 2,756 | 4,706 |
| AXI4-Stream | 8 | 2 | 8 | - | 16 | 0 | 3,274 | 5,476 |
| AXI4-Stream | 8 | 3 | 1 | - | 3 | 0 | 456 | 722 |

*Table 1-3:* **Virtex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| AXI4-Stream | 8 | 3 | 2 | - | 6 | 0 | 749 | 1,168 |
| AXI4-Stream | 8 | 3 | 3 | - | 9 | 0 | 1,108 | 1,964 |
| AXI4-Stream | 8 | 3 | 4 | - | 12 | 0 | 1,454 | 2,592 |
| AXI4-Stream | 8 | 3 | 5 | - | 15 | 0 | 2,098 | 3,823 |
| AXI4-Stream | 8 | 3 | 6 | - | 18 | 0 | 2,684 | 4,651 |
| AXI4-Stream | 8 | 3 | 7 | - | 21 | 0 | 3,268 | 5,716 |
| AXI4-Stream | 8 | 3 | 8 | - | 24 | 0 | 3,774 | 6,640 |
| AXI4-Stream | 10 | 2 | 1 | - | 2 | 0 | 442 | 694 |
| AXI4-Stream | 10 | 2 | 2 | - | 4 | 0 | 698 | 1,108 |
| AXI4-Stream | 10 | 2 | 3 | - | 6 | 0 | 1,064 | 1,857 |
| AXI4-Stream | 10 | 2 | 4 | - | 8 | 0 | 1,443 | 2,459 |
| AXI4-Stream | 10 | 2 | 5 | - | 10 | 0 | 2,063 | 3,636 |
| AXI4-Stream | 10 | 2 | 6 | - | 12 | 0 | 2,610 | 4,437 |
| AXI4-Stream | 10 | 2 | 7 | - | 14 | 0 | 3,163 | 5,473 |
| AXI4-Stream | 10 | 2 | 8 | - | 16 | 0 | 3,679 | 6,375 |
| AXI4-Stream | 10 | 3 | 1 | - | 3 | 0 | 472 | 815 |
| AXI4-Stream | 10 | 3 | 2 | - | 6 | 0 | 810 | 1,332 |
| AXI4-Stream | 10 | 3 | 3 | - | 9 | 0 | 1,292 | 2,273 |
| AXI4-Stream | 10 | 3 | 4 | - | 12 | 0 | 1,720 | 3,010 |
| AXI4-Stream | 10 | 3 | 5 | - | 15 | 0 | 2,424 | 4,484 |
| AXI4-Stream | 10 | 3 | 6 | - | 18 | 0 | 3,116 | 5,461 |
| AXI4-Stream | 10 | 3 | 7 | - | 21 | 0 | 3,744 | 6,729 |
| AXI4-Stream | 10 | 3 | 8 | - | 24 | 0 | 4,336 | 7,824 |
| AXI4-Stream | 12 | 2 | 1 | - | 2 | 0 | 469 | 763 |
| AXI4-Stream | 12 | 2 | 2 | - | 4 | 0 | 778 | 1,229 |
| AXI4-Stream | 12 | 2 | 3 | - | 6 | 0 | 1,211 | 2,086 |
| AXI4-Stream | 12 | 2 | 4 | - | 8 | 0 | 1,607 | 2,770 |
| AXI4-Stream | 12 | 2 | 5 | - | 10 | 0 | 2,284 | 4,132 |
| AXI4-Stream | 12 | 2 | 6 | - | 12 | 0 | 2,883 | 5,047 |
| AXI4-Stream | 12 | 2 | 7 | - | 14 | 0 | 3,568 | 6,238 |
| AXI4-Stream | 12 | 2 | 8 | - | 16 | 0 | 4,189 | 7,274 |
| AXI4-Stream | 12 | 3 | 1 | - | 3 | 0 | 519 | 907 |

*Table 1-3:* **Virtex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| AXI4-Stream | 12 | 3 | 2 | - | 6 | 0 | 898 | 1,497 |
| AXI4-Stream | 12 | 3 | 3 | - | 9 | 0 | 1,362 | 2,582 |
| AXI4-Stream | 12 | 3 | 4 | - | 12 | 0 | 1,884 | 3,428 |
| AXI4-Stream | 12 | 3 | 5 | - | 15 | 0 | 2,860 | 5,146 |
| AXI4-Stream | 12 | 3 | 6 | - | 18 | 0 | 3,479 | 6,272 |
| AXI4-Stream | 12 | 3 | 7 | - | 21 | 0 | 4,140 | 7,743 |
| AXI4-Stream | 12 | 3 | 8 | - | 24 | 0 | 4,991 | 9,008 |
| XSVI | 8 | 2 | 1 | - | 2 | 0 | 413 | 636 |
| XSVI | 8 | 2 | 2 | - | 4 | 0 | 627 | 1,037 |
| XSVI | 8 | 2 | 3 | - | 6 | 0 | 987 | 1,627 |
| XSVI | 8 | 2 | 4 | - | 8 | 0 | 1,279 | 2,100 |
| XSVI | 8 | 2 | 5 | - | 10 | 0 | 1,864 | 3,050 |
| XSVI | 8 | 2 | 6 | - | 12 | 0 | 2,386 | 3,679 |
| XSVI | 8 | 2 | 7 | - | 14 | 0 | 2,847 | 4,499 |
| XSVI | 8 | 2 | 8 | - | 16 | 0 | 3,336 | 5,178 |
| XSVI | 8 | 3 | 1 | - | 3 | 0 | 467 | 812 |
| XSVI | 8 | 3 | 2 | - | 6 | 0 | 755 | 1,271 |
| XSVI | 8 | 3 | 3 | - | 9 | 0 | 1,136 | 2,024 |
| XSVI | 8 | 3 | 4 | - | 12 | 0 | 1,550 | 2,611 |
| XSVI | 8 | 3 | 5 | - | 15 | 0 | 2,400 | 3,817 |
| XSVI | 8 | 3 | 6 | - | 18 | 0 | 2,926 | 4,597 |
| XSVI | 8 | 3 | 7 | - | 21 | 0 | 3,538 | 5,615 |
| XSVI | 8 | 3 | 8 | - | 24 | 0 | 4,109 | 6,449 |
| XSVI | 10 | 2 | 1 | - | 2 | 0 | 453 | 764 |
| XSVI | 10 | 2 | 2 | - | 4 | 0 | 703 | 1,185 |
| XSVI | 10 | 2 | 3 | - | 6 | 0 | 1,096 | 1,884 |
| XSVI | 10 | 2 | 4 | - | 8 | 0 | 1,496 | 2,439 |
| XSVI | 10 | 2 | 5 | - | 10 | 0 | 2,171 | 3,577 |
| XSVI | 10 | 2 | 6 | - | 12 | 0 | 2,750 | 4,324 |
| XSVI | 10 | 2 | 7 | - | 14 | 0 | 3,334 | 5,303 |
| XSVI | 10 | 2 | 8 | - | 16 | 0 | 3,875 | 6,111 |
| XSVI | 10 | 3 | 1 | - | 3 | 0 | 536 | 933 |

*Table 1-3:* **Virtex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| XSVI | 10 | 3 | 2 | - | 6 | 0 | 881 | 1,475 |
| XSVI | 10 | 3 | 3 | - | 9 | 0 | 1,375 | 2,378 |
| XSVI | 10 | 3 | 4 | - | 12 | 0 | 1,849 | 3,075 |
| XSVI | 10 | 3 | 5 | - | 15 | 0 | 2,863 | 4,531 |
| XSVI | 10 | 3 | 6 | - | 18 | 0 | 3,500 | 5,463 |
| XSVI | 10 | 3 | 7 | - | 21 | 0 | 4,229 | 6,695 |
| XSVI | 10 | 3 | 8 | - | 24 | 0 | 4,855 | 8,280 |
| XSVI | 12 | 2 | 1 | - | 2 | 0 | 500 | 853 |
| XSVI | 12 | 2 | 2 | - | 4 | 0 | 776 | 1,332 |
| XSVI | 12 | 2 | 3 | - | 6 | 0 | 1,242 | 2,139 |
| XSVI | 12 | 2 | 4 | - | 8 | 0 | 1,690 | 2,778 |
| XSVI | 12 | 2 | 5 | - | 10 | 0 | 2,631 | 4,106 |
| XSVI | 12 | 2 | 6 | - | 12 | 0 | 3,207 | 4,968 |
| XSVI | 12 | 2 | 7 | - | 14 | 0 | 3,809 | 6,104 |
| XSVI | 12 | 2 | 8 | - | 16 | 0 | 4,546 | 7,042 |
| XSVI | 12 | 3 | 1 | - | 3 | 0 | 550 | 1,055 |
| XSVI | 12 | 3 | 2 | - | 6 | 0 | 975 | 1,680 |
| XSVI | 12 | 3 | 3 | - | 9 | 0 | 1,544 | 2,731 |
| XSVI | 12 | 3 | 4 | - | 12 | 0 | 2,121 | 3,540 |
| XSVI | 12 | 3 | 5 | - | 15 | 0 | 3,406 | 5,249 |
| XSVI | 12 | 3 | 6 | - | 18 | 0 | 4,151 | 6,333 |
| XSVI | 12 | 3 | 7 | - | 21 | 0 | 4,933 | 8,349 |
| XSVI | 12 | 3 | 8 | - | 24 | 0 | 5,657 | 8,937 |

*Table 1-4:* **Kintex-7 Resource Estimates**

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 8 | 2 | 1 | 320 | 2 | 2 | 1,157 | 1,350 |
| Graphics Controller | 8 | 2 | 1 | 4096 | 2 | 3 | 1,193 | 1,362 |
| Graphics Controller | 8 | 2 | 2 | 320 | 4 | 4 | 2,164 | 2,441 |
| Graphics Controller | 8 | 2 | 2 | 4096 | 4 | 6 | 2,183 | 2,466 |

*Table 1-4:* **Kintex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 8 | 2 | 3 | 320 | 6 | 6 | 3,336 | 3,732 |
| Graphics Controller | 8 | 2 | 3 | 4096 | 6 | 9 | 3,297 | 3,769 |
| Graphics Controller | 8 | 2 | 4 | 320 | 8 | 8 | 4,339 | 4,941 |
| Graphics Controller | 8 | 2 | 4 | 4096 | 8 | 12 | 4,406 | 4,991 |
| Graphics Controller | 8 | 2 | 5 | 320 | 10 | 10 | 5,623 | 6,596 |
| Graphics Controller | 8 | 2 | 5 | 4096 | 10 | 15 | 5,610 | 6,658 |
| Graphics Controller | 8 | 2 | 6 | 320 | 12 | 12 | 6,834 | 7,953 |
| Graphics Controller | 8 | 2 | 6 | 4096 | 12 | 18 | 6,850 | 8,027 |
| Graphics Controller | 8 | 2 | 7 | 320 | 14 | 14 | 8,021 | 9,907 |
| Graphics Controller | 8 | 2 | 7 | 4096 | 14 | 21 | 8,158 | 9,997 |
| Graphics Controller | 8 | 2 | 8 | 320 | 16 | 16 | 9,253 | 11,417 |
| Graphics Controller | 8 | 2 | 8 | 4096 | 16 | 24 | 9,275 | 11,520 |
| Graphics Controller | 8 | 3 | 1 | 320 | 3 | 2 | 1,215 | 1,458 |
| Graphics Controller | 8 | 3 | 1 | 4096 | 3 | 3 | 1,211 | 1,470 |
| Graphics Controller | 8 | 3 | 2 | 320 | 6 | 4 | 2,340 | 2,647 |
| Graphics Controller | 8 | 3 | 2 | 4096 | 6 | 6 | 2,288 | 2,672 |
| Graphics Controller | 8 | 3 | 3 | 320 | 9 | 6 | 3,411 | 4,084 |
| Graphics Controller | 8 | 3 | 3 | 4096 | 9 | 9 | 3,455 | 4,121 |
| Graphics Controller | 8 | 3 | 4 | 320 | 12 | 8 | 4,686 | 5,406 |
| Graphics Controller | 8 | 3 | 4 | 4096 | 12 | 12 | 4,674 | 5,455 |
| Graphics Controller | 8 | 3 | 5 | 320 | 15 | 10 | 5,953 | 7,297 |
| Graphics Controller | 8 | 3 | 5 | 4096 | 15 | 15 | 6,003 | 7,359 |
| Graphics Controller | 8 | 3 | 6 | 320 | 18 | 12 | 7,309 | 9,194 |
| Graphics Controller | 8 | 3 | 6 | 4096 | 18 | 18 | 7,307 | 9,272 |
| Graphics Controller | 8 | 3 | 7 | 320 | 21 | 14 | 8,508 | 11,011 |
| Graphics Controller | 8 | 3 | 7 | 4096 | 21 | 21 | 8,632 | 11,100 |
| Graphics Controller | 8 | 3 | 8 | 320 | 24 | 16 | 9,835 | 12,684 |
| Graphics Controller | 8 | 3 | 8 | 4096 | 24 | 24 | 10,003 | 12,787 |
| Graphics Controller | 10 | 2 | 1 | 320 | 2 | 2 | 1,176 | 1,457 |
| Graphics Controller | 10 | 2 | 1 | 4096 | 2 | 3 | 1,171 | 1,469 |
| Graphics Controller | 10 | 2 | 2 | 320 | 4 | 4 | 2,229 | 2,638 |
| Graphics Controller | 10 | 2 | 2 | 4096 | 4 | 6 | 2,259 | 2,663 |

*Table 1-4:* **Kintex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 10 | 2 | 3 | 320 | 6 | 6 | 3,383 | 4,061 |
| Graphics Controller | 10 | 2 | 3 | 4096 | 6 | 9 | 3,361 | 4,098 |
| Graphics Controller | 10 | 2 | 4 | 320 | 8 | 8 | 4,550 | 5,384 |
| Graphics Controller | 10 | 2 | 4 | 4096 | 8 | 12 | 4,556 | 5,433 |
| Graphics Controller | 10 | 2 | 5 | 320 | 10 | 10 | 5,698 | 7,250 |
| Graphics Controller | 10 | 2 | 5 | 4096 | 10 | 15 | 5,795 | 7,312 |
| Graphics Controller | 10 | 2 | 6 | 320 | 12 | 12 | 7,067 | 9,125 |
| Graphics Controller | 10 | 2 | 6 | 4096 | 12 | 18 | 7,041 | 9,202 |
| Graphics Controller | 10 | 2 | 7 | 320 | 14 | 14 | 8,254 | 10,932 |
| Graphics Controller | 10 | 2 | 7 | 4096 | 14 | 21 | 8,379 | 11,022 |
| Graphics Controller | 10 | 2 | 8 | 320 | 16 | 16 | 9,581 | 12,612 |
| Graphics Controller | 10 | 2 | 8 | 4096 | 16 | 24 | 9,687 | 12,715 |
| Graphics Controller | 10 | 3 | 1 | 320 | 3 | 2 | 1,237 | 1,587 |
| Graphics Controller | 10 | 3 | 1 | 4096 | 3 | 3 | 1,220 | 1,600 |
| Graphics Controller | 10 | 3 | 2 | 320 | 6 | 4 | 2,362 | 2,884 |
| Graphics Controller | 10 | 3 | 2 | 4096 | 6 | 6 | 2,327 | 2,909 |
| Graphics Controller | 10 | 3 | 3 | 320 | 9 | 6 | 3,511 | 4,482 |
| Graphics Controller | 10 | 3 | 3 | 4096 | 9 | 9 | 3,597 | 4,520 |
| Graphics Controller | 10 | 3 | 4 | 320 | 12 | 8 | 4,747 | 5,941 |
| Graphics Controller | 10 | 3 | 4 | 4096 | 12 | 12 | 4,857 | 5,991 |
| Graphics Controller | 10 | 3 | 5 | 320 | 15 | 10 | 6,228 | 8,097 |
| Graphics Controller | 10 | 3 | 5 | 4096 | 15 | 15 | 6,287 | 8,159 |
| Graphics Controller | 10 | 3 | 6 | 320 | 18 | 12 | 7,614 | 10,214 |
| Graphics Controller | 10 | 3 | 6 | 4096 | 18 | 18 | 7,553 | 10,292 |
| Graphics Controller | 10 | 3 | 7 | 320 | 21 | 14 | 8,913 | 12,272 |
| Graphics Controller | 10 | 3 | 7 | 4096 | 21 | 21 | 9,056 | 12,362 |
| Graphics Controller | 10 | 3 | 8 | 320 | 24 | 16 | 10,420 | 14,149 |
| Graphics Controller | 10 | 3 | 8 | 4096 | 24 | 24 | 10,466 | 14,252 |
| Graphics Controller | 12 | 2 | 1 | 320 | 2 | 2 | 1,275 | 1,533 |
| Graphics Controller | 12 | 2 | 1 | 4096 | 2 | 3 | 1,231 | 1,545 |
| Graphics Controller | 12 | 2 | 2 | 320 | 4 | 4 | 2,360 | 2,775 |
| Graphics Controller | 12 | 2 | 2 | 4096 | 4 | 6 | 2,314 | 2,800 |

*Table 1-4:* **Kintex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 12 | 2 | 3 | 320 | 6 | 6 | 3,443 | 4,299 |
| Graphics Controller | 12 | 2 | 3 | 4096 | 6 | 9 | 3,554 | 4,336 |
| Graphics Controller | 12 | 2 | 4 | 320 | 8 | 8 | 4,690 | 5,705 |
| Graphics Controller | 12 | 2 | 4 | 4096 | 8 | 12 | 4,677 | 5,755 |
| Graphics Controller | 12 | 2 | 5 | 320 | 10 | 10 | 6,087 | 7,751 |
| Graphics Controller | 12 | 2 | 5 | 4096 | 10 | 15 | 6,177 | 7,814 |
| Graphics Controller | 12 | 2 | 6 | 320 | 12 | 12 | 7,418 | 9,790 |
| Graphics Controller | 12 | 2 | 6 | 4096 | 12 | 18 | 7,541 | 9,867 |
| Graphics Controller | 12 | 2 | 7 | 320 | 14 | 14 | 8,739 | 11,762 |
| Graphics Controller | 12 | 2 | 7 | 4096 | 14 | 21 | 8,868 | 11,852 |
| Graphics Controller | 12 | 2 | 8 | 320 | 16 | 16 | 10,255 | 13,584 |
| Graphics Controller | 12 | 2 | 8 | 4096 | 16 | 24 | 10,222 | 13,687 |
| Graphics Controller | 12 | 3 | 1 | 320 | 3 | 2 | 1,269 | 1,685 |
| Graphics Controller | 12 | 3 | 1 | 4096 | 3 | 3 | 1,291 | 1,698 |
| Graphics Controller | 12 | 3 | 2 | 320 | 6 | 4 | 2,454 | 3,059 |
| Graphics Controller | 12 | 3 | 2 | 4096 | 6 | 6 | 2,435 | 3,084 |
| Graphics Controller | 12 | 3 | 3 | 320 | 9 | 6 | 3,747 | 4,790 |
| Graphics Controller | 12 | 3 | 3 | 4096 | 9 | 9 | 3,752 | 4,828 |
| Graphics Controller | 12 | 3 | 4 | 320 | 12 | 8 | 4,882 | 6,354 |
| Graphics Controller | 12 | 3 | 4 | 4096 | 12 | 12 | 5,070 | 6,404 |
| Graphics Controller | 12 | 3 | 5 | 320 | 15 | 10 | 6,561 | 9,146 |
| Graphics Controller | 12 | 3 | 5 | 4096 | 15 | 15 | 6,691 | 9,211 |
| Graphics Controller | 12 | 3 | 6 | 320 | 18 | 12 | 8,018 | 11,069 |
| Graphics Controller | 12 | 3 | 6 | 4096 | 18 | 18 | 8,094 | 11,145 |
| Graphics Controller | 12 | 3 | 7 | 320 | 21 | 14 | 9,564 | 13,335 |
| Graphics Controller | 12 | 3 | 7 | 4096 | 21 | 21 | 9,432 | 13,425 |
| Graphics Controller | 12 | 3 | 8 | 320 | 24 | 16 | 10,953 | 15,380 |
| Graphics Controller | 12 | 3 | 8 | 4096 | 24 | 24 | 11,135 | 15,485 |
| AXI4-Stream | 8 | 2 | 1 | - | 2 | 0 | 432 | 625 |
| AXI4-Stream | 8 | 2 | 2 | - | 4 | 0 | 638 | 1,033 |
| AXI4-Stream | 8 | 2 | 3 | - | 6 | 0 | 945 | 1,629 |
| AXI4-Stream | 8 | 2 | 4 | - | 8 | 0 | 1,263 | 2,148 |

*Table 1-4:* **Kintex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| AXI4-Stream | 8 | 2 | 5 | - | 10 | 0 | 1,841 | 3,141 |
| AXI4-Stream | 8 | 2 | 6 | - | 12 | 0 | 2,182 | 4,050 |
| AXI4-Stream | 8 | 2 | 7 | - | 14 | 0 | 2,704 | 4,977 |
| AXI4-Stream | 8 | 2 | 8 | - | 16 | 0 | 3,096 | 5,476 |
| AXI4-Stream | 8 | 3 | 1 | - | 3 | 0 | 399 | 722 |
| AXI4-Stream | 8 | 3 | 2 | - | 6 | 0 | 700 | 1,228 |
| AXI4-Stream | 8 | 3 | 3 | - | 9 | 0 | 1,057 | 1,964 |
| AXI4-Stream | 8 | 3 | 4 | - | 12 | 0 | 1,428 | 2,592 |
| AXI4-Stream | 8 | 3 | 5 | - | 15 | 0 | 2,090 | 3,823 |
| AXI4-Stream | 8 | 3 | 6 | - | 18 | 0 | 2,534 | 4,651 |
| AXI4-Stream | 8 | 3 | 7 | - | 21 | 0 | 3,102 | 5,716 |
| AXI4-Stream | 8 | 3 | 8 | - | 24 | 0 | 3,577 | 6,640 |
| AXI4-Stream | 10 | 2 | 1 | - | 2 | 0 | 486 | 694 |
| AXI4-Stream | 10 | 2 | 2 | - | 4 | 0 | 719 | 1,163 |
| AXI4-Stream | 10 | 2 | 3 | - | 6 | 0 | 1,084 | 1,857 |
| AXI4-Stream | 10 | 2 | 4 | - | 8 | 0 | 1,444 | 2,459 |
| AXI4-Stream | 10 | 2 | 5 | - | 10 | 0 | 1,955 | 3,636 |
| AXI4-Stream | 10 | 2 | 6 | - | 12 | 0 | 2,602 | 4,437 |
| AXI4-Stream | 10 | 2 | 7 | - | 14 | 0 | 3,083 | 5,473 |
| AXI4-Stream | 10 | 2 | 8 | - | 16 | 0 | 3,491 | 6,375 |
| AXI4-Stream | 10 | 3 | 1 | - | 3 | 0 | 541 | 815 |
| AXI4-Stream | 10 | 3 | 2 | - | 6 | 0 | 812 | 1,403 |
| AXI4-Stream | 10 | 3 | 3 | - | 9 | 0 | 1,229 | 2,273 |
| AXI4-Stream | 10 | 3 | 4 | - | 12 | 0 | 1,563 | 3,010 |
| AXI4-Stream | 10 | 3 | 5 | - | 15 | 0 | 2,338 | 4,484 |
| AXI4-Stream | 10 | 3 | 6 | - | 18 | 0 | 3,000 | 5,461 |
| AXI4-Stream | 10 | 3 | 7 | - | 21 | 0 | 3,593 | 6,729 |
| AXI4-Stream | 10 | 3 | 8 | - | 24 | 0 | 4,131 | 7,824 |
| AXI4-Stream | 12 | 2 | 1 | - | 2 | 0 | 444 | 763 |
| AXI4-Stream | 12 | 2 | 2 | - | 4 | 0 | 742 | 1,292 |
| AXI4-Stream | 12 | 2 | 3 | - | 6 | 0 | 1,153 | 2,086 |
| AXI4-Stream | 12 | 2 | 4 | - | 8 | 0 | 1,547 | 2,770 |

*Table 1-4:* **Kintex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| AXI4-Stream | 12 | 2 | 5 | - | 10 | 0 | 2,250 | 4,132 |
| AXI4-Stream | 12 | 2 | 6 | - | 12 | 0 | 2,851 | 5,047 |
| AXI4-Stream | 12 | 2 | 7 | - | 14 | 0 | 3,426 | 6,238 |
| AXI4-Stream | 12 | 2 | 8 | - | 16 | 0 | 3,895 | 7,274 |
| AXI4-Stream | 12 | 3 | 1 | - | 3 | 0 | 551 | 907 |
| AXI4-Stream | 12 | 3 | 2 | - | 6 | 0 | 831 | 1,579 |
| AXI4-Stream | 12 | 3 | 3 | - | 9 | 0 | 1,348 | 2,582 |
| AXI4-Stream | 12 | 3 | 4 | - | 12 | 0 | 1,800 | 3,428 |
| AXI4-Stream | 12 | 3 | 5 | - | 15 | 0 | 2,652 | 5,146 |
| AXI4-Stream | 12 | 3 | 6 | - | 18 | 0 | 3,269 | 6,272 |
| AXI4-Stream | 12 | 3 | 7 | - | 21 | 0 | 3,959 | 7,743 |
| AXI4-Stream | 12 | 3 | 8 | - | 24 | 0 | 4,738 | 9,008 |
| XSVI | 8 | 2 | 1 | - | 2 | 0 | 388 | 676 |
| XSVI | 8 | 2 | 2 | - | 4 | 0 | 637 | 1,037 |
| XSVI | 8 | 2 | 3 | - | 6 | 0 | 955 | 1,627 |
| XSVI | 8 | 2 | 4 | - | 8 | 0 | 1,277 | 2,100 |
| XSVI | 8 | 2 | 5 | - | 10 | 0 | 1,853 | 3,050 |
| XSVI | 8 | 2 | 6 | - | 12 | 0 | 2,269 | 3,679 |
| XSVI | 8 | 2 | 7 | - | 14 | 0 | 2,737 | 4,499 |
| XSVI | 8 | 2 | 8 | - | 16 | 0 | 3,137 | 5,177 |
| XSVI | 8 | 3 | 1 | - | 3 | 0 | 409 | 812 |
| XSVI | 8 | 3 | 2 | - | 6 | 0 | 727 | 1,271 |
| XSVI | 8 | 3 | 3 | - | 9 | 0 | 1,097 | 2,025 |
| XSVI | 8 | 3 | 4 | - | 12 | 0 | 1,475 | 2,611 |
| XSVI | 8 | 3 | 5 | - | 15 | 0 | 2,279 | 3,817 |
| XSVI | 8 | 3 | 6 | - | 18 | 0 | 2,849 | 4,595 |
| XSVI | 8 | 3 | 7 | - | 21 | 0 | 3,273 | 5,616 |
| XSVI | 8 | 3 | 8 | - | 24 | 0 | 3,896 | 6,448 |
| XSVI | 10 | 2 | 1 | - | 2 | 0 | 416 | 764 |
| XSVI | 10 | 2 | 2 | - | 4 | 0 | 705 | 1,185 |
| XSVI | 10 | 2 | 3 | - | 6 | 0 | 1,165 | 1,884 |
| XSVI | 10 | 2 | 4 | - | 8 | 0 | 1,478 | 2,440 |

*Table 1-4:* **Kintex-7 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| XSVI | 10 | 2 | 5 | - | 10 | 0 | 2,181 | 3,577 |
| XSVI | 10 | 2 | 6 | - | 12 | 0 | 2,722 | 4,322 |
| XSVI | 10 | 2 | 7 | - | 14 | 0 | 3,243 | 5,302 |
| XSVI | 10 | 2 | 8 | - | 16 | 0 | 3,760 | 6,109 |
| XSVI | 10 | 3 | 1 | - | 3 | 0 | 579 | 933 |
| XSVI | 10 | 3 | 2 | - | 6 | 0 | 844 | 1,476 |
| XSVI | 10 | 3 | 3 | - | 9 | 0 | 1,290 | 2,378 |
| XSVI | 10 | 3 | 4 | - | 12 | 0 | 1,744 | 3,076 |
| XSVI | 10 | 3 | 5 | - | 15 | 0 | 2,722 | 4,533 |
| XSVI | 10 | 3 | 6 | - | 18 | 0 | 3,407 | 5,464 |
| XSVI | 10 | 3 | 7 | - | 21 | 0 | 3,878 | 6,695 |
| XSVI | 10 | 3 | 8 | - | 24 | 0 | 4,615 | 7,695 |
| XSVI | 12 | 2 | 1 | - | 2 | 0 | 465 | 853 |
| XSVI | 12 | 2 | 2 | - | 4 | 0 | 756 | 1,332 |
| XSVI | 12 | 2 | 3 | - | 6 | 0 | 1,184 | 2,141 |
| XSVI | 12 | 2 | 4 | - | 8 | 0 | 1,581 | 2,778 |
| XSVI | 12 | 2 | 5 | - | 10 | 0 | 2,479 | 4,105 |
| XSVI | 12 | 2 | 6 | - | 12 | 0 | 3,127 | 4,966 |
| XSVI | 12 | 2 | 7 | - | 14 | 0 | 3,660 | 6,105 |
| XSVI | 12 | 2 | 8 | - | 16 | 0 | 4,418 | 7,039 |
| XSVI | 12 | 3 | 1 | - | 3 | 0 | 510 | 1,055 |
| XSVI | 12 | 3 | 2 | - | 6 | 0 | 906 | 1,680 |
| XSVI | 12 | 3 | 3 | - | 9 | 0 | 1,492 | 2,732 |
| XSVI | 12 | 3 | 4 | - | 12 | 0 | 1,953 | 3,540 |
| XSVI | 12 | 3 | 5 | - | 15 | 0 | 3,146 | 5,249 |
| XSVI | 12 | 3 | 6 | - | 18 | 0 | 4,004 | 6,333 |
| XSVI | 12 | 3 | 7 | - | 21 | 0 | 4,521 | 7,773 |
| XSVI | 12 | 3 | 8 | - | 24 | 0 | 5,423 | 8,938 |

*Table 1-5:* **Virtex-6 Resource Estimates**

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 8 | 2 | 1 | 320 | 2 | 2 | 1,490 | 1,302 |
| Graphics Controller | 8 | 2 | 1 | 4096 | 2 | 3 | 1,428 | 1,315 |
| Graphics Controller | 8 | 2 | 2 | 320 | 4 | 4 | 2,743 | 2,356 |
| Graphics Controller | 8 | 2 | 2 | 4096 | 4 | 6 | 2,762 | 2,379 |
| Graphics Controller | 8 | 2 | 3 | 320 | 6 | 6 | 3,970 | 3,593 |
| Graphics Controller | 8 | 2 | 3 | 4096 | 6 | 9 | 4,079 | 3,629 |
| Graphics Controller | 8 | 2 | 4 | 320 | 8 | 8 | 5,410 | 4,745 |
| Graphics Controller | 8 | 2 | 4 | 4096 | 8 | 12 | 5,428 | 4,799 |
| Graphics Controller | 8 | 2 | 5 | 320 | 10 | 10 | 7,091 | 6,326 |
| Graphics Controller | 8 | 2 | 5 | 4096 | 10 | 15 | 7,147 | 6,386 |
| Graphics Controller | 8 | 2 | 6 | 320 | 12 | 12 | 8,612 | 7,614 |
| Graphics Controller | 8 | 2 | 6 | 4096 | 12 | 18 | 8,711 | 8,033 |
| Graphics Controller | 8 | 2 | 7 | 320 | 14 | 14 | 9,760 | 9,505 |
| Graphics Controller | 8 | 2 | 7 | 4096 | 14 | 21 | 9,900 | 10,006 |
| Graphics Controller | 8 | 2 | 8 | 320 | 16 | 16 | 11,583 | 11,422 |
| Graphics Controller | 8 | 2 | 8 | 4096 | 16 | 24 | 11,687 | 11,527 |
| Graphics Controller | 8 | 3 | 1 | 320 | 3 | 2 | 1,567 | 1,404 |
| Graphics Controller | 8 | 3 | 1 | 4096 | 3 | 3 | 1,483 | 1,419 |
| Graphics Controller | 8 | 3 | 2 | 320 | 6 | 4 | 2,876 | 2,549 |
| Graphics Controller | 8 | 3 | 2 | 4096 | 6 | 6 | 2,874 | 2,572 |
| Graphics Controller | 8 | 3 | 3 | 320 | 9 | 6 | 4,277 | 3,921 |
| Graphics Controller | 8 | 3 | 3 | 4096 | 9 | 9 | 4,236 | 3,961 |
| Graphics Controller | 8 | 3 | 4 | 320 | 12 | 8 | 5,798 | 5,411 |
| Graphics Controller | 8 | 3 | 4 | 4096 | 12 | 12 | 5,839 | 5,460 |
| Graphics Controller | 8 | 3 | 5 | 320 | 15 | 10 | 7,360 | 7,305 |
| Graphics Controller | 8 | 3 | 5 | 4096 | 15 | 15 | 7,313 | 7,681 |
| Graphics Controller | 8 | 3 | 6 | 320 | 18 | 12 | 9,149 | 9,204 |
| Graphics Controller | 8 | 3 | 6 | 4096 | 18 | 18 | 9,211 | 9,277 |
| Graphics Controller | 8 | 3 | 7 | 320 | 21 | 14 | 10,452 | 10,541 |
| Graphics Controller | 8 | 3 | 7 | 4096 | 21 | 21 | 10,591 | 11,108 |
| Graphics Controller | 8 | 3 | 8 | 320 | 24 | 16 | 12,300 | 12,698 |

*Table 1-5:* **Virtex-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 8 | 3 | 8 | 4096 | 24 | 24 | 12,484 | 12,795 |
| Graphics Controller | 10 | 2 | 1 | 320 | 2 | 2 | 1,559 | 1,406 |
| Graphics Controller | 10 | 2 | 1 | 4096 | 2 | 3 | 1,455 | 1,419 |
| Graphics Controller | 10 | 2 | 2 | 320 | 4 | 4 | 2,823 | 2,547 |
| Graphics Controller | 10 | 2 | 2 | 4096 | 4 | 6 | 2,763 | 2,570 |
| Graphics Controller | 10 | 2 | 3 | 320 | 6 | 6 | 4,175 | 3,912 |
| Graphics Controller | 10 | 2 | 3 | 4096 | 6 | 9 | 4,047 | 3,948 |
| Graphics Controller | 10 | 2 | 4 | 320 | 8 | 8 | 5,596 | 5,175 |
| Graphics Controller | 10 | 2 | 4 | 4096 | 8 | 12 | 5,683 | 5,223 |
| Graphics Controller | 10 | 2 | 5 | 320 | 10 | 10 | 7,216 | 7,260 |
| Graphics Controller | 10 | 2 | 5 | 4096 | 10 | 15 | 7,469 | 7,317 |
| Graphics Controller | 10 | 2 | 6 | 320 | 12 | 12 | 8,879 | 8,756 |
| Graphics Controller | 10 | 2 | 6 | 4096 | 12 | 18 | 8,655 | 9,208 |
| Graphics Controller | 10 | 2 | 7 | 320 | 14 | 14 | 9,807 | 10,489 |
| Graphics Controller | 10 | 2 | 7 | 4096 | 14 | 21 | 9,900 | 11,031 |
| Graphics Controller | 10 | 2 | 8 | 320 | 16 | 16 | 11,453 | 12,619 |
| Graphics Controller | 10 | 2 | 8 | 4096 | 16 | 24 | 11,605 | 12,722 |
| Graphics Controller | 10 | 3 | 1 | 320 | 3 | 2 | 1,586 | 1,529 |
| Graphics Controller | 10 | 3 | 1 | 4096 | 3 | 3 | 1,460 | 1,542 |
| Graphics Controller | 10 | 3 | 2 | 320 | 6 | 4 | 2,981 | 2,777 |
| Graphics Controller | 10 | 3 | 2 | 4096 | 6 | 6 | 2,902 | 2,800 |
| Graphics Controller | 10 | 3 | 3 | 320 | 9 | 6 | 4,372 | 4,304 |
| Graphics Controller | 10 | 3 | 3 | 4096 | 9 | 9 | 4,394 | 4,344 |
| Graphics Controller | 10 | 3 | 4 | 320 | 12 | 8 | 6,037 | 5,945 |
| Graphics Controller | 10 | 3 | 4 | 4096 | 12 | 12 | 5,993 | 5,995 |
| Graphics Controller | 10 | 3 | 5 | 320 | 15 | 10 | 7,543 | 8,104 |
| Graphics Controller | 10 | 3 | 5 | 4096 | 15 | 15 | 8,050 | 8,524 |
| Graphics Controller | 10 | 3 | 6 | 320 | 18 | 12 | 9,522 | 9,778 |
| Graphics Controller | 10 | 3 | 6 | 4096 | 18 | 18 | 9,578 | 10,304 |
| Graphics Controller | 10 | 3 | 7 | 320 | 21 | 14 | 10,903 | 11,749 |
| Graphics Controller | 10 | 3 | 7 | 4096 | 21 | 21 | 10,603 | 12,368 |
| Graphics Controller | 10 | 3 | 8 | 320 | 24 | 16 | 13,058 | 14,162 |

*Table 1-5:* **Virtex-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 10 | 3 | 8 | 4096 | 24 | 24 | 12,692 | 14,260 |
| Graphics Controller | 12 | 2 | 1 | 320 | 2 | 2 | 1,590 | 1,477 |
| Graphics Controller | 12 | 2 | 1 | 4096 | 2 | 3 | 1,471 | 1,490 |
| Graphics Controller | 12 | 2 | 2 | 320 | 4 | 4 | 2,880 | 2,673 |
| Graphics Controller | 12 | 2 | 2 | 4096 | 4 | 6 | 2,876 | 2,695 |
| Graphics Controller | 12 | 2 | 3 | 320 | 6 | 6 | 4,255 | 4,132 |
| Graphics Controller | 12 | 2 | 3 | 4096 | 6 | 9 | 4,374 | 4,168 |
| Graphics Controller | 12 | 2 | 4 | 320 | 8 | 8 | 5,786 | 5,709 |
| Graphics Controller | 12 | 2 | 4 | 4096 | 8 | 12 | 5,979 | 5,759 |
| Graphics Controller | 12 | 2 | 5 | 320 | 10 | 10 | 7,649 | 7,761 |
| Graphics Controller | 12 | 2 | 5 | 4096 | 10 | 15 | 7,654 | 8,161 |
| Graphics Controller | 12 | 2 | 6 | 320 | 12 | 12 | 9,276 | 9,369 |
| Graphics Controller | 12 | 2 | 6 | 4096 | 12 | 18 | 9,160 | 9,872 |
| Graphics Controller | 12 | 2 | 7 | 320 | 14 | 14 | 10,625 | 11,259 |
| Graphics Controller | 12 | 2 | 7 | 4096 | 14 | 21 | 10,258 | 11,861 |
| Graphics Controller | 12 | 2 | 8 | 320 | 16 | 16 | 12,205 | 13,590 |
| Graphics Controller | 12 | 2 | 8 | 4096 | 16 | 24 | 12,478 | 13,695 |
| Graphics Controller | 12 | 3 | 1 | 320 | 3 | 2 | 1,755 | 1,620 |
| Graphics Controller | 12 | 3 | 1 | 4096 | 3 | 3 | 1,541 | 1,633 |
| Graphics Controller | 12 | 3 | 2 | 320 | 6 | 4 | 3,044 | 2,938 |
| Graphics Controller | 12 | 3 | 2 | 4096 | 6 | 6 | 3,039 | 2,960 |
| Graphics Controller | 12 | 3 | 3 | 320 | 9 | 6 | 4,584 | 4,794 |
| Graphics Controller | 12 | 3 | 3 | 4096 | 9 | 9 | 4,608 | 4,628 |
| Graphics Controller | 12 | 3 | 4 | 320 | 12 | 8 | 6,277 | 6,362 |
| Graphics Controller | 12 | 3 | 4 | 4096 | 12 | 12 | 6,264 | 6,408 |
| Graphics Controller | 12 | 3 | 5 | 320 | 15 | 10 | 8,051 | 8,752 |
| Graphics Controller | 12 | 3 | 5 | 4096 | 15 | 15 | 8,141 | 9,222 |
| Graphics Controller | 12 | 3 | 6 | 320 | 18 | 12 | 10,060 | 10,567 |
| Graphics Controller | 12 | 3 | 6 | 4096 | 18 | 18 | 10,232 | 11,156 |
| Graphics Controller | 12 | 3 | 7 | 320 | 21 | 14 | 11,383 | 13,346 |
| Graphics Controller | 12 | 3 | 7 | 4096 | 21 | 14 | 11,383 | 13,346 |
| Graphics Controller | 12 | 3 | 8 | 320 | 24 | 16 | 13,745 | 15,394 |

*Table 1-5:* **Virtex-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 12 | 3 | 8 | 4096 | 24 | 24 | 13,324 | 15,492 |
| AXI4-Stream | 8 | 2 | 1 | - | 2 | 0 | 420 | 625 |
| AXI4-Stream | 8 | 2 | 2 | - | 4 | 0 | 653 | 1,033 |
| AXI4-Stream | 8 | 2 | 3 | - | 6 | 0 | 1,011 | 1,629 |
| AXI4-Stream | 8 | 2 | 4 | - | 8 | 0 | 1,307 | 2,148 |
| AXI4-Stream | 8 | 2 | 5 | - | 10 | 0 | 1,823 | 3,141 |
| AXI4-Stream | 8 | 2 | 6 | - | 12 | 0 | 2,262 | 4,050 |
| AXI4-Stream | 8 | 2 | 7 | - | 14 | 0 | 2,720 | 4,977 |
| AXI4-Stream | 8 | 2 | 8 | - | 16 | 0 | 3,171 | 5,476 |
| AXI4-Stream | 8 | 3 | 1 | - | 3 | 0 | 457 | 722 |
| AXI4-Stream | 8 | 3 | 2 | - | 6 | 0 | 728 | 1,228 |
| AXI4-Stream | 8 | 3 | 3 | - | 9 | 0 | 1,109 | 1,964 |
| AXI4-Stream | 8 | 3 | 4 | - | 12 | 0 | 1,521 | 2,592 |
| AXI4-Stream | 8 | 3 | 5 | - | 15 | 0 | 2,109 | 3,823 |
| AXI4-Stream | 8 | 3 | 6 | - | 18 | 0 | 2,613 | 4,931 |
| AXI4-Stream | 8 | 3 | 7 | - | 21 | 0 | 3,139 | 5,716 |
| AXI4-Stream | 8 | 3 | 8 | - | 24 | 0 | 3,687 | 6,640 |
| AXI4-Stream | 10 | 2 | 1 | - | 2 | 0 | 459 | 694 |
| AXI4-Stream | 10 | 2 | 2 | - | 4 | 0 | 740 | 1,163 |
| AXI4-Stream | 10 | 2 | 3 | - | 6 | 0 | 1,049 | 1,857 |
| AXI4-Stream | 10 | 2 | 4 | - | 8 | 0 | 1,488 | 2,459 |
| AXI4-Stream | 10 | 2 | 5 | - | 10 | 0 | 2,069 | 3,636 |
| AXI4-Stream | 10 | 2 | 6 | - | 12 | 0 | 2,611 | 4,701 |
| AXI4-Stream | 10 | 2 | 7 | - | 14 | 0 | 3,103 | 5,473 |
| AXI4-Stream | 10 | 2 | 8 | - | 16 | 0 | 3,567 | 6,375 |
| AXI4-Stream | 10 | 3 | 1 | - | 3 | 0 | 487 | 815 |
| AXI4-Stream | 10 | 3 | 2 | - | 6 | 0 | 787 | 1,403 |
| AXI4-Stream | 10 | 3 | 3 | - | 9 | 0 | 1,268 | 2,273 |
| AXI4-Stream | 10 | 3 | 4 | - | 12 | 0 | 1,701 | 3,010 |
| AXI4-Stream | 10 | 3 | 5 | - | 15 | 0 | 2,489 | 4,484 |
| AXI4-Stream | 10 | 3 | 6 | - | 18 | 0 | 3,106 | 5,461 |
| AXI4-Stream | 10 | 3 | 7 | - | 21 | 0 | 3,608 | 6,729 |

*Table 1-5:* **Virtex-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| AXI4-Stream | 10 | 3 | 8 | - | 24 | 0 | 4,233 | 7,824 |
| AXI4-Stream | 12 | 2 | 1 | - | 2 | 0 | 496 | 763 |
| AXI4-Stream | 12 | 2 | 2 | - | 4 | 0 | 763 | 1,292 |
| AXI4-Stream | 12 | 2 | 3 | - | 6 | 0 | 1,200 | 2,086 |
| AXI4-Stream | 12 | 2 | 4 | - | 8 | 0 | 1,521 | 2,770 |
| AXI4-Stream | 12 | 2 | 5 | - | 10 | 0 | 2,317 | 4,132 |
| AXI4-Stream | 12 | 2 | 6 | - | 12 | 0 | 2,911 | 5,047 |
| AXI4-Stream | 12 | 2 | 7 | - | 14 | 0 | 3,505 | 6,238 |
| AXI4-Stream | 12 | 2 | 8 | - | 16 | 0 | 4,055 | 7,274 |
| AXI4-Stream | 12 | 3 | 1 | - | 3 | 0 | 500 | 907 |
| AXI4-Stream | 12 | 3 | 2 | - | 6 | 0 | 883 | 1,579 |
| AXI4-Stream | 12 | 3 | 3 | - | 9 | 0 | 1,385 | 2,582 |
| AXI4-Stream | 12 | 3 | 4 | - | 12 | 0 | 1,835 | 3,428 |
| AXI4-Stream | 12 | 3 | 5 | - | 15 | 0 | 2,751 | 5,146 |
| AXI4-Stream | 12 | 3 | 6 | - | 18 | 0 | 3,379 | 6,272 |
| AXI4-Stream | 12 | 3 | 7 | - | 21 | 0 | 4,063 | 7,743 |
| AXI4-Stream | 12 | 3 | 8 | - | 24 | 0 | 4,927 | 9,008 |
| XSVI | 8 | 2 | 1 | - | 2 | 0 | 428 | 676 |
| XSVI | 8 | 2 | 2 | - | 4 | 0 | 626 | 1,037 |
| XSVI | 8 | 2 | 3 | - | 6 | 0 | 962 | 1,626 |
| XSVI | 8 | 2 | 4 | - | 8 | 0 | 1,233 | 2,100 |
| XSVI | 8 | 2 | 5 | - | 10 | 0 | 1,894 | 3,051 |
| XSVI | 8 | 2 | 6 | - | 12 | 0 | 2,316 | 3,680 |
| XSVI | 8 | 2 | 7 | - | 14 | 0 | 2,777 | 4,499 |
| XSVI | 8 | 2 | 8 | - | 16 | 0 | 3,315 | 5,178 |
| XSVI | 8 | 3 | 1 | - | 3 | 0 | 489 | 811 |
| XSVI | 8 | 3 | 2 | - | 6 | 0 | 778 | 1,271 |
| XSVI | 8 | 3 | 3 | - | 9 | 0 | 1,173 | 2,024 |
| XSVI | 8 | 3 | 4 | - | 12 | 0 | 1,517 | 2,612 |
| XSVI | 8 | 3 | 5 | - | 15 | 0 | 2,362 | 3,818 |
| XSVI | 8 | 3 | 6 | - | 18 | 0 | 2,889 | 4,597 |
| XSVI | 8 | 3 | 7 | - | 21 | 0 | 3,483 | 5,617 |

*Table 1-5:* **Virtex-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| XSVI | 8 | 3 | 8 | - | 24 | 0 | 3,958 | 6,445 |
| XSVI | 10 | 2 | 1 | - | 2 | 0 | 469 | 764 |
| XSVI | 10 | 2 | 2 | - | 4 | 0 | 744 | 1,185 |
| XSVI | 10 | 2 | 3 | - | 6 | 0 | 1,111 | 1,884 |
| XSVI | 10 | 2 | 4 | - | 8 | 0 | 1,449 | 2,440 |
| XSVI | 10 | 2 | 5 | - | 10 | 0 | 2,216 | 3,578 |
| XSVI | 10 | 2 | 6 | - | 12 | 0 | 2,758 | 4,326 |
| XSVI | 10 | 2 | 7 | - | 14 | 0 | 3,352 | 5,304 |
| XSVI | 10 | 2 | 8 | - | 16 | 0 | 3,770 | 6,107 |
| XSVI | 10 | 3 | 1 | - | 3 | 0 | 556 | 933 |
| XSVI | 10 | 3 | 2 | - | 6 | 0 | 849 | 1,475 |
| XSVI | 10 | 3 | 3 | - | 9 | 0 | 1,443 | 2,378 |
| XSVI | 10 | 3 | 4 | - | 12 | 0 | 1,767 | 3,076 |
| XSVI | 10 | 3 | 5 | - | 15 | 0 | 2,845 | 4,533 |
| XSVI | 10 | 3 | 6 | - | 18 | 0 | 3,542 | 5,466 |
| XSVI | 10 | 3 | 7 | - | 21 | 0 | 4,228 | 6,695 |
| XSVI | 10 | 3 | 8 | - | 24 | 0 | 4,743 | 8,271 |
| XSVI | 12 | 2 | 1 | - | 2 | 0 | 491 | 853 |
| XSVI | 12 | 2 | 2 | - | 4 | 0 | 820 | 1,332 |
| XSVI | 12 | 2 | 3 | - | 6 | 0 | 1,201 | 2,140 |
| XSVI | 12 | 2 | 4 | - | 8 | 0 | 1,645 | 2,779 |
| XSVI | 12 | 2 | 5 | - | 10 | 0 | 2,631 | 4,106 |
| XSVI | 12 | 2 | 6 | - | 12 | 0 | 3,128 | 4,968 |
| XSVI | 12 | 2 | 7 | - | 14 | 0 | 3,751 | 6,105 |
| XSVI | 12 | 2 | 8 | - | 16 | 0 | 4,510 | 7,039 |
| XSVI | 12 | 3 | 1 | - | 3 | 0 | 543 | 1,054 |
| XSVI | 12 | 3 | 2 | - | 6 | 0 | 942 | 1,680 |
| XSVI | 12 | 3 | 3 | - | 9 | 0 | 1,493 | 2,731 |
| XSVI | 12 | 3 | 4 | - | 12 | 0 | 2,054 | 3,540 |
| XSVI | 12 | 3 | 5 | - | 15 | 0 | 3,267 | 5,249 |
| XSVI | 12 | 3 | 6 | - | 18 | 0 | 4,015 | 6,333 |

*Table 1-5:* **Virtex-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| XSVI | 12 | 3 | 7 | - | 21 | 0 | 4,688 | 7,774 |
| XSVI | 12 | 3 | 8 | - | 24 | 0 | 5,558 | 9,611 |

*Table 1-6:* **Spartan-6 Resource Estimates**

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 8 | 2 | 1 | 320 | 2 | 2 | 1,572 | 1,426 |
| Graphics Controller | 8 | 2 | 1 | 4096 | 2 | 5 | 1,576 | 1,384 |
| Graphics Controller | 8 | 2 | 2 | 320 | 4 | 4 | 2,738 | 2,692 |
| Graphics Controller | 8 | 2 | 2 | 4096 | 4 | 10 | 3,110 | 2,617 |
| Graphics Controller | 8 | 2 | 3 | 320 | 6 | 6 | 4,162 | 4,117 |
| Graphics Controller | 8 | 2 | 3 | 4096 | 6 | 15 | 4,695 | 3,996 |
| Graphics Controller | 8 | 2 | 4 | 320 | 8 | 8 | 5,599 | 5,461 |
| Graphics Controller | 8 | 2 | 4 | 4096 | 8 | 20 | 6,161 | 5,293 |
| Graphics Controller | 8 | 2 | 5 | 320 | 10 | 10 | 7,884 | 7,273 |
| Graphics Controller | 8 | 2 | 5 | 4096 | 10 | 25 | 8,005 | 7,032 |
| Graphics Controller | 8 | 2 | 6 | 320 | 12 | 12 | 9,450 | 8,401 |
| Graphics Controller | 8 | 2 | 6 | 4096 | 12 | 30 | 9,565 | 8,477 |
| Graphics Controller | 8 | 2 | 7 | 320 | 14 | 14 | 11,238 | 10,027 |
| Graphics Controller | 8 | 2 | 7 | 4096 | 14 | 35 | 11,257 | 10,115 |
| Graphics Controller | 8 | 2 | 8 | 320 | 16 | 16 | 12,799 | 11,531 |
| Graphics Controller | 8 | 2 | 8 | 4096 | 16 | 40 | 13,005 | 11,632 |
| Graphics Controller | 8 | 3 | 1 | 320 | 3 | 2 | 1,582 | 1,535 |
| Graphics Controller | 8 | 3 | 1 | 4096 | 3 | 5 | 1,659 | 1,486 |
| Graphics Controller | 8 | 3 | 2 | 320 | 6 | 4 | 2,888 | 2,910 |
| Graphics Controller | 8 | 3 | 2 | 4096 | 6 | 10 | 3,219 | 2,823 |
| Graphics Controller | 8 | 3 | 3 | 320 | 9 | 6 | 4,318 | 4,491 |
| Graphics Controller | 8 | 3 | 3 | 4096 | 9 | 15 | 4,840 | 4,351 |
| Graphics Controller | 8 | 3 | 4 | 320 | 12 | 8 | 6,524 | 5,957 |
| Graphics Controller | 8 | 3 | 4 | 4096 | 12 | 20 | 6,493 | 5,757 |
| Graphics Controller | 8 | 3 | 5 | 320 | 15 | 10 | 8,473 | 8,020 |

*Table 1-6:* **Spartan-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 8 | 3 | 5 | 4096 | 15 | 25 | 8,379 | 7,734 |
| Graphics Controller | 8 | 3 | 6 | 320 | 18 | 12 | 10,063 | 9,242 |
| Graphics Controller | 8 | 3 | 6 | 4096 | 18 | 30 | 10,201 | 9,317 |
| Graphics Controller | 8 | 3 | 7 | 320 | 21 | 14 | 11,895 | 11,061 |
| Graphics Controller | 8 | 3 | 7 | 4096 | 21 | 35 | 12,027 | 11,149 |
| Graphics Controller | 8 | 2 | 1 | 320 | 2 | 2 | 1,572 | 1,426 |
| Graphics Controller | 8 | 2 | 1 | 4096 | 2 | 5 | 1,576 | 1,384 |
| Graphics Controller | 8 | 2 | 2 | 320 | 4 | 4 | 2,738 | 2,692 |
| Graphics Controller | 8 | 2 | 2 | 4096 | 4 | 10 | 3,110 | 2,617 |
| Graphics Controller | 8 | 2 | 3 | 320 | 6 | 6 | 4,162 | 4,117 |
| Graphics Controller | 8 | 2 | 3 | 4096 | 6 | 15 | 4,695 | 3,996 |
| Graphics Controller | 8 | 2 | 4 | 320 | 8 | 8 | 5,599 | 5,461 |
| Graphics Controller | 8 | 2 | 4 | 4096 | 8 | 20 | 6,161 | 5,293 |
| Graphics Controller | 8 | 2 | 5 | 320 | 10 | 10 | 7,884 | 7,273 |
| Graphics Controller | 8 | 2 | 5 | 4096 | 10 | 25 | 8,005 | 7,032 |
| Graphics Controller | 8 | 2 | 6 | 320 | 12 | 12 | 9,450 | 8,401 |
| Graphics Controller | 8 | 2 | 6 | 4096 | 12 | 30 | 9,565 | 8,477 |
| Graphics Controller | 8 | 2 | 7 | 320 | 14 | 14 | 11,238 | 10,027 |
| Graphics Controller | 8 | 2 | 7 | 4096 | 14 | 35 | 11,257 | 10,115 |
| Graphics Controller | 8 | 2 | 8 | 320 | 16 | 16 | 12,799 | 11,531 |
| Graphics Controller | 8 | 2 | 8 | 4096 | 16 | 40 | 13,005 | 11,632 |
| Graphics Controller | 8 | 3 | 1 | 320 | 3 | 2 | 1,582 | 1,535 |
| Graphics Controller | 8 | 3 | 1 | 4096 | 3 | 5 | 1,659 | 1,486 |
| Graphics Controller | 8 | 3 | 2 | 320 | 6 | 4 | 2,888 | 2,910 |
| Graphics Controller | 8 | 3 | 2 | 4096 | 6 | 10 | 3,219 | 2,823 |
| Graphics Controller | 8 | 3 | 3 | 320 | 9 | 6 | 4,318 | 4,491 |
| Graphics Controller | 8 | 3 | 3 | 4096 | 9 | 15 | 4,840 | 4,351 |
| Graphics Controller | 8 | 3 | 4 | 320 | 12 | 8 | 6,524 | 5,957 |
| Graphics Controller | 8 | 3 | 4 | 4096 | 12 | 20 | 6,493 | 5,757 |
| Graphics Controller | 8 | 3 | 5 | 320 | 15 | 10 | 8,473 | 8,020 |
| Graphics Controller | 8 | 3 | 5 | 4096 | 15 | 25 | 8,379 | 7,734 |
| Graphics Controller | 8 | 3 | 6 | 320 | 18 | 12 | 10,063 | 9,242 |

*Table 1-6:* **Spartan-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 8 | 3 | 6 | 4096 | 18 | 30 | 10,201 | 9,317 |
| Graphics Controller | 8 | 3 | 7 | 320 | 21 | 14 | 11,895 | 11,061 |
| Graphics Controller | 8 | 3 | 7 | 4096 | 21 | 35 | 12,027 | 11,149 |
| Graphics Controller | 8 | 3 | 8 | 320 | 24 | 16 | 12,815 | 12,706 |
| Graphics Controller | 8 | 3 | 8 | 4096 | 24 | 40 | 13,119 | 12,803 |
| Graphics Controller | 10 | 2 | 1 | 320 | 2 | 2 | 1,614 | 1,533 |
| Graphics Controller | 10 | 2 | 1 | 4096 | 2 | 5 | 1,624 | 1,488 |
| Graphics Controller | 10 | 2 | 2 | 320 | 4 | 4 | 2,843 | 2,895 |
| Graphics Controller | 10 | 2 | 2 | 4096 | 4 | 10 | 3,160 | 2,814 |
| Graphics Controller | 10 | 2 | 3 | 320 | 6 | 6 | 4,252 | 4,457 |
| Graphics Controller | 10 | 2 | 3 | 4096 | 6 | 15 | 4,733 | 4,327 |
| Graphics Controller | 10 | 2 | 4 | 320 | 8 | 8 | 5,747 | 5,921 |
| Graphics Controller | 10 | 2 | 4 | 4096 | 8 | 20 | 6,351 | 5,738 |
| Graphics Controller | 10 | 2 | 5 | 320 | 10 | 10 | 8,238 | 7,953 |
| Graphics Controller | 10 | 2 | 5 | 4096 | 10 | 25 | 8,285 | 7,686 |
| Graphics Controller | 10 | 2 | 6 | 320 | 12 | 12 | 10,037 | 9,612 |
| Graphics Controller | 10 | 2 | 6 | 4096 | 12 | 30 | 9,851 | 9,274 |
| Graphics Controller | 10 | 2 | 7 | 320 | 14 | 14 | 11,577 | 11,010 |
| Graphics Controller | 10 | 2 | 7 | 4096 | 14 | 35 | 11,543 | 11,098 |
| Graphics Controller | 10 | 2 | 8 | 320 | 16 | 16 | 13,235 | 12,670 |
| Graphics Controller | 10 | 2 | 8 | 4096 | 16 | 40 | 13,403 | 12,771 |
| Graphics Controller | 10 | 3 | 1 | 320 | 3 | 2 | 1,675 | 1,663 |
| Graphics Controller | 10 | 3 | 1 | 4096 | 3 | 5 | 1,713 | 1,611 |
| Graphics Controller | 10 | 3 | 2 | 320 | 6 | 4 | 3,012 | 3,157 |
| Graphics Controller | 10 | 3 | 2 | 4096 | 6 | 10 | 3,313 | 3,060 |
| Graphics Controller | 10 | 3 | 3 | 320 | 9 | 6 | 4,600 | 4,904 |
| Graphics Controller | 10 | 3 | 3 | 4096 | 9 | 15 | 5,138 | 4,748 |
| Graphics Controller | 10 | 3 | 4 | 320 | 12 | 8 | 6,882 | 6,514 |
| Graphics Controller | 10 | 3 | 4 | 4096 | 12 | 20 | 6,808 | 6,291 |
| Graphics Controller | 10 | 3 | 5 | 320 | 15 | 10 | 8,846 | 8,857 |
| Graphics Controller | 10 | 3 | 5 | 4096 | 15 | 25 | 8,792 | 8,534 |
| Graphics Controller | 10 | 3 | 6 | 320 | 18 | 12 | 10,790 | 10,701 |

*Table 1-6:* **Spartan-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 10 | 3 | 6 | 4096 | 18 | 30 | 10,657 | 10,290 |
| Graphics Controller | 10 | 3 | 7 | 320 | 21 | 14 | 12,352 | 12,260 |
| Graphics Controller | 10 | 3 | 7 | 4096 | 21 | 35 | 12,068 | 12,337 |
| Graphics Controller | 10 | 3 | 8 | 320 | 24 | 16 | 14,180 | 14,577 |
| Graphics Controller | 10 | 3 | 8 | 4096 | 24 | 40 | 13,547 | 14,669 |
| Graphics Controller | 12 | 2 | 1 | 320 | 2 | 2 | 1,629 | 1,610 |
| Graphics Controller | 12 | 2 | 1 | 4096 | 2 | 5 | 1,663 | 1,559 |
| Graphics Controller | 12 | 2 | 2 | 320 | 4 | 4 | 2,970 | 3,043 |
| Graphics Controller | 12 | 2 | 2 | 4096 | 4 | 10 | 3,258 | 2,951 |
| Graphics Controller | 12 | 2 | 3 | 320 | 6 | 6 | 4,497 | 4,713 |
| Graphics Controller | 12 | 2 | 3 | 4096 | 6 | 15 | 4,857 | 4,567 |
| Graphics Controller | 12 | 2 | 4 | 320 | 8 | 8 | 5,854 | 6,266 |
| Graphics Controller | 12 | 2 | 4 | 4096 | 8 | 20 | 6,559 | 6,055 |
| Graphics Controller | 12 | 2 | 5 | 320 | 10 | 10 | 8,547 | 8,493 |
| Graphics Controller | 12 | 2 | 5 | 4096 | 10 | 25 | 8,376 | 8,188 |
| Graphics Controller | 12 | 2 | 6 | 320 | 12 | 12 | 10,165 | 10,276 |
| Graphics Controller | 12 | 2 | 6 | 4096 | 12 | 30 | 10,390 | 9,887 |
| Graphics Controller | 12 | 2 | 7 | 320 | 14 | 14 | 12,004 | 11,780 |
| Graphics Controller | 12 | 2 | 7 | 4096 | 14 | 35 | 12,105 | 11,868 |
| Graphics Controller | 12 | 2 | 8 | 320 | 16 | 16 | 13,728 | 13,569 |
| Graphics Controller | 12 | 2 | 8 | 4096 | 16 | 40 | 13,881 | 13,670 |
| Graphics Controller | 12 | 3 | 1 | 320 | 3 | 2 | 1,685 | 1,762 |
| Graphics Controller | 12 | 3 | 1 | 4096 | 3 | 5 | 1,735 | 1,702 |
| Graphics Controller | 12 | 3 | 2 | 320 | 6 | 4 | 3,083 | 3,347 |
| Graphics Controller | 12 | 3 | 2 | 4096 | 6 | 10 | 3,422 | 3,236 |
| Graphics Controller | 12 | 3 | 3 | 320 | 9 | 6 | 4,827 | 5,235 |
| Graphics Controller | 12 | 3 | 3 | 4096 | 9 | 15 | 5,240 | 5,054 |
| Graphics Controller | 12 | 3 | 4 | 320 | 12 | 8 | 7,027 | 6,964 |
| Graphics Controller | 12 | 3 | 4 | 4096 | 12 | 20 | 6,949 | 6,704 |
| Graphics Controller | 12 | 3 | 5 | 320 | 15 | 10 | 9,218 | 9,554 |
| Graphics Controller | 12 | 3 | 5 | 4096 | 15 | 25 | 9,134 | 9,182 |
| Graphics Controller | 12 | 3 | 6 | 320 | 18 | 12 | 11,411 | 11,917 |

*Table 1-6:* **Spartan-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| Graphics Controller | 12 | 3 | 6 | 4096 | 18 | 30 | 11,248 | 11,079 |
| Graphics Controller | 12 | 3 | 7 | 320 | 21 | 14 | 12,663 | 13,695 |
| Graphics Controller | 12 | 3 | 7 | 4096 | 21 | 35 | 12,210 | 13,782 |
| Graphics Controller | 12 | 3 | 8 | 320 | 24 | 16 | 14,108 | 15,819 |
| Graphics Controller | 12 | 3 | 8 | 4096 | 24 | 40 | 13,848 | 15,897 |
| AXI4-Stream | 8 | 2 | 1 | - | 2 | 0 | 449 | 652 |
| AXI4-Stream | 8 | 2 | 2 | - | 4 | 0 | 714 | 1,033 |
| AXI4-Stream | 8 | 2 | 3 | - | 6 | 0 | 998 | 1,630 |
| AXI4-Stream | 8 | 2 | 4 | - | 8 | 0 | 1,334 | 2,148 |
| AXI4-Stream | 8 | 2 | 5 | - | 10 | 0 | 2,024 | 3,312 |
| AXI4-Stream | 8 | 2 | 6 | - | 12 | 0 | 2,542 | 4,049 |
| AXI4-Stream | 8 | 2 | 7 | - | 14 | 0 | 3,210 | 4,969 |
| AXI4-Stream | 8 | 2 | 8 | - | 16 | 0 | 3,847 | 5,797 |
| AXI4-Stream | 8 | 3 | 1 | - | 3 | 0 | 465 | 754 |
| AXI4-Stream | 8 | 3 | 2 | - | 6 | 0 | 769 | 1,228 |
| AXI4-Stream | 8 | 3 | 3 | - | 9 | 0 | 1,202 | 1,964 |
| AXI4-Stream | 8 | 3 | 4 | - | 12 | 0 | 1,596 | 2,592 |
| AXI4-Stream | 8 | 3 | 5 | - | 15 | 0 | 2,384 | 4,035 |
| AXI4-Stream | 8 | 3 | 6 | - | 18 | 0 | 2,987 | 4,924 |
| AXI4-Stream | 8 | 3 | 7 | - | 21 | 0 | 3,608 | 6,050 |
| AXI4-Stream | 8 | 3 | 8 | - | 24 | 0 | 4,213 | 7,044 |
| AXI4-Stream | 10 | 2 | 1 | - | 2 | 0 | 490 | 725 |
| AXI4-Stream | 10 | 2 | 2 | - | 4 | 0 | 814 | 1,162 |
| AXI4-Stream | 10 | 2 | 3 | - | 6 | 0 | 1,200 | 1,856 |
| AXI4-Stream | 10 | 2 | 4 | - | 8 | 0 | 1,589 | 2,460 |
| AXI4-Stream | 10 | 2 | 5 | - | 10 | 0 | 2,343 | 3,838 |
| AXI4-Stream | 10 | 2 | 6 | - | 12 | 0 | 3,149 | 4,694 |
| AXI4-Stream | 10 | 2 | 7 | - | 14 | 0 | 3,627 | 5,786 |
| AXI4-Stream | 10 | 2 | 8 | - | 16 | 0 | 5,508 | 6,830 |
| AXI4-Stream | 10 | 3 | 1 | - | 3 | 0 | 484 | 852 |
| AXI4-Stream | 10 | 3 | 2 | - | 6 | 0 | 862 | 1,403 |
| AXI4-Stream | 10 | 3 | 3 | - | 9 | 0 | 1,356 | 2,273 |

*Table 1-6:* **Spartan-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| AXI4-Stream | 10 | 3 | 4 | - | 12 | 0 | 1,817 | 3,011 |
| AXI4-Stream | 10 | 3 | 5 | - | 15 | 0 | 2,866 | 4,739 |
| AXI4-Stream | 10 | 3 | 6 | - | 18 | 0 | 3,568 | 5,790 |
| AXI4-Stream | 10 | 3 | 7 | - | 21 | 0 | 3,965 | 7,124 |
| AXI4-Stream | 10 | 3 | 8 | - | 24 | 0 | 4,982 | 8,303 |
| AXI4-Stream | 12 | 2 | 1 | - | 2 | 0 | 484 | 797 |
| AXI4-Stream | 12 | 2 | 2 | - | 4 | 0 | 846 | 1,292 |
| AXI4-Stream | 12 | 2 | 3 | - | 6 | 0 | 1,288 | 2,086 |
| AXI4-Stream | 12 | 2 | 4 | - | 8 | 0 | 1,738 | 2,771 |
| AXI4-Stream | 12 | 2 | 5 | - | 10 | 0 | 2,640 | 4,360 |
| AXI4-Stream | 12 | 2 | 6 | - | 12 | 0 | 3,540 | 5,344 |
| AXI4-Stream | 12 | 2 | 7 | - | 14 | 0 | 4,342 | 6,600 |
| AXI4-Stream | 12 | 2 | 8 | - | 16 | 0 | 6,389 | 7,793 |
| AXI4-Stream | 12 | 3 | 1 | - | 3 | 0 | 562 | 950 |
| AXI4-Stream | 12 | 3 | 2 | - | 6 | 0 | 912 | 1,578 |
| AXI4-Stream | 12 | 3 | 3 | - | 9 | 0 | 1,536 | 2,583 |
| AXI4-Stream | 12 | 3 | 4 | - | 12 | 0 | 2,006 | 3,429 |
| AXI4-Stream | 12 | 3 | 5 | - | 15 | 0 | 3,202 | 5,443 |
| AXI4-Stream | 12 | 3 | 6 | - | 18 | 0 | 3,896 | 6,656 |
| AXI4-Stream | 12 | 3 | 7 | - | 21 | 0 | 4,721 | 8,204 |
| AXI4-Stream | 12 | 3 | 8 | - | 24 | 0 | 6,081 | 9,575 |
| XSVI | 8 | 2 | 1 | - | 2 | 0 | 478 | 716 |
| XSVI | 8 | 2 | 2 | - | 4 | 0 | 714 | 1,099 |
| XSVI | 8 | 2 | 3 | - | 6 | 0 | 1,673 | 1,960 |
| XSVI | 8 | 2 | 4 | - | 8 | 0 | 2,042 | 2,692 |
| XSVI | 8 | 2 | 5 | - | 10 | 0 | 1,982 | 3,051 |
| XSVI | 8 | 2 | 6 | - | 12 | 0 | 2,401 | 3,679 |
| XSVI | 8 | 2 | 7 | - | 14 | 0 | 3,015 | 4,795 |
| XSVI | 8 | 2 | 8 | - | 16 | 0 | 5,212 | 6,159 |
| XSVI | 8 | 3 | 1 | - | 3 | 0 | 516 | 866 |
| XSVI | 8 | 3 | 2 | - | 6 | 0 | 845 | 1,356 |
| XSVI | 8 | 3 | 3 | - | 9 | 0 | 1,950 | 2,538 |

*Table 1-6:* **Spartan-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| XSVI | 8 | 3 | 4 | - | 12 | 0 | 2,830 | 3,354 |
| XSVI | 8 | 3 | 5 | - | 15 | 0 | 2,480 | 3,818 |
| XSVI | 8 | 3 | 6 | - | 18 | 0 | 2,978 | 4,926 |
| XSVI | 8 | 3 | 7 | - | 21 | 0 | 5,263 | 6,806 |
| XSVI | 8 | 3 | 8 | - | 24 | 0 | 7,029 | 7,643 |
| XSVI | 10 | 2 | 1 | - | 2 | 0 | 534 | 810 |
| XSVI | 10 | 2 | 2 | - | 4 | 0 | 788 | 1,261 |
| XSVI | 10 | 2 | 3 | - | 6 | 0 | 1,912 | 2,256 |
| XSVI | 10 | 2 | 4 | - | 8 | 0 | 2,594 | 3,165 |
| XSVI | 10 | 2 | 5 | - | 10 | 0 | 2,302 | 3,578 |
| XSVI | 10 | 2 | 6 | - | 12 | 0 | 3,034 | 4,745 |
| XSVI | 10 | 2 | 7 | - | 14 | 0 | 3,502 | 5,663 |
| XSVI | 10 | 2 | 8 | - | 16 | 0 | 6,041 | 7,262 |
| XSVI | 10 | 3 | 1 | - | 3 | 0 | 631 | 996 |
| XSVI | 10 | 3 | 2 | - | 6 | 0 | 954 | 1,580 |
| XSVI | 10 | 3 | 3 | - | 9 | 0 | 2,389 | 2,904 |
| XSVI | 10 | 3 | 4 | - | 12 | 0 | 3,676 | 4,045 |
| XSVI | 10 | 3 | 5 | - | 15 | 0 | 2,952 | 4,853 |
| XSVI | 10 | 3 | 6 | - | 18 | 0 | 5,722 | 6,544 |
| XSVI | 10 | 3 | 7 | - | 21 | 0 | 7,274 | 7,950 |
| XSVI | 10 | 3 | 8 | - | 24 | 0 | 8,205 | 9,196 |
| XSVI | 12 | 2 | 1 | - | 2 | 0 | 549 | 907 |
| XSVI | 12 | 2 | 2 | - | 4 | 0 | 917 | 1,424 |
| XSVI | 12 | 2 | 3 | - | 6 | 0 | 1,244 | 2,280 |
| XSVI | 12 | 2 | 4 | - | 8 | 0 | 3,026 | 3,617 |
| XSVI | 12 | 2 | 5 | - | 10 | 0 | 2,663 | 4,107 |
| XSVI | 12 | 2 | 6 | - | 12 | 0 | 5,273 | 6,022 |
| XSVI | 12 | 2 | 7 | - | 14 | 0 | 6,543 | 7,344 |
| XSVI | 12 | 2 | 8 | - | 16 | 0 | 7,351 | 8,367 |
| XSVI | 12 | 3 | 1 | - | 3 | 0 | 672 | 1,130 |
| XSVI | 12 | 3 | 2 | - | 6 | 0 | 1,097 | 1,808 |
| XSVI | 12 | 3 | 3 | - | 9 | 0 | 2,671 | 3,355 |

*Table 1-6:* **Spartan-6 Resource Estimates *(Cont'd)***

| Layer Type | Data Width | Channels | Layers | Maximum Screen Width | XtremeDSP Slices | BRAM | LUTs | FFs |
|---|---|---|---|---|---|---|---|---|
| XSVI | 12 | 3 | 4 | - | 12 | 0 | 3,875 | 4,583 |
| XSVI | 12 | 3 | 5 | - | 15 | 0 | 3,536 | 5,628 |
| XSVI | 12 | 3 | 6 | - | 18 | 0 | 6,687 | 7,591 |
| XSVI | 12 | 3 | 7 | - | 21 | 0 | 8,275 | 9,189 |
| XSVI | 12 | 3 | 8 | - | 24 | 0 | 8,394 | 10,310 |

XILINX®

Chapter 2

# Core Interfaces and Register Space

This chapter provides detailed descriptions for each interface. In addition, detailed information about configuration and control registers is included.

## Port Descriptions

### Core Interfaces

#### AXI4-Stream Interface

The Video On-Screen Display core uses an AXI4-Stream interface to connect to the AXI VDMA and other Xilinx Video IP. The AXI VDMA core provides access to external memory, and registers that allow the user to specify the location in memory of the various layer data buffers that the OSD core accesses. The OSD core provides registers for configuring the placement, size and transparency of each video layer.

#### Processor Interface

There are many video systems that use an integrated processor system to dynamically control the parameters within the system. This is important when several independent image processing cores are integrated into a single FPGA. The Video On-Screen Display core can be configured with one of two interfaces: an EDK pCore Interface (AXI4-Lite) or a General Purpose Processor Interface (GPP).

### Common I/O Signals

The EDK pCore interface and the General Purpose Processor interface share a number of the same input/output (I/O) signals. The signals that both interfaces share are specified in Table 2-1.

*Table 2-1:* **Common Port Descriptions**

| Port Name | Dir | Width | Description |
|---|---|---|---|
| clk | I | 1 | CORE CLOCK<br>Core clock (active High edge). |
| ce | I | 1 | CLOCK ENABLE<br>Used to halt processing and hold current values. |

*Table 2-1:* **Common Port Descriptions *(Cont'd)***

| Port Name | Dir | Width | Description |
|---|---|---|---|
| **Input Xilinx Streaming Video Interface** | | | |
| vblank_in | I | 1 | INPUT VERTICAL BLANKING PERIOD SIGNAL<br>Denotes the video lines during which no active video pixel is present. Can be active High or active Low polarity. Used internally for synchronization. The known polarity of this input must be set in the OSD Control register before enabling the OSD.<br>The minimum requirements for vblank are based on the use of the graphics controller. If a user is using the graphics controller of the OSD, then the vblank must be held for 1 line. If the user is not using the graphics controller, then the vblank can be as short as 1 clock cycle. This one cycle can be pulsed before an hblank pulse. |
| vsync_in | I | 1 | INPUT VERTICAL SYNCHRONIZATION SIGNAL<br>This is the vertical synchronization pulse. It is not used internally to the OSD. It is delayed and presented on the `vsync_out` output aligned to the output data. |
| hblank_in | I | 1 | INPUT HORIZONTAL BLANKING PERIOD SIGNAL<br>Denotes the cycles during which no active video pixel is present. Can be active High or active Low polarity. Used internally for synchronization. The known polarity of this input must be set in the OSD Control register before enabling the OSD.<br>The minimum requirement for hblank is 1 clock cycle. |
| hsync_in | I | 1 | INPUT HORIZONTAL SYNCHRONIZATION SIGNAL<br>This is the horizontal synchronization pulse. It is not used internally to the OSD. It is delayed and presented on the `hsync_out` output. |
| active_video_in | I | 1 | INPUT ACTIVE VIDEO<br>Denotes cycles during which active video is present. Can be active High or active Low. It is delayed and presented on the active_video_out output. |

*Table 2-1:* **Common Port Descriptions *(Cont'd)***

| Port Name | Dir | Width | Description |
|---|---|---|---|
| active_chroma_in | I | 1 | INPUT ACTIVE CHROMA<br>Denotes cycles during which valid chroma is present on the video_data_in bus. It is not used internally to the OSD. It is delayed and presented on the `active_chroma_out` output. |
| video_data_in | I | [(C_NUM_DATA_CHANNELS+ C_ALPHA_CHANNEL_EN)* C_DATA_WIDTH-1:0] | INPUT VIDEO DATA<br>Streaming input data. This data can be configured to be routed to one or more layers, which allows enabling graphics layers to be blended to streaming data without the use of external memory. |
| **Slave AXI4-Stream Interfaces(4)** | | | |
| s<*LAYER_NUM*>_axis_tdata | I | [n-1: 0](1) | AXI4- STREAM DATA IN<br>Input AXI4-Stream data.<br>Input layer data for layers set to External AXIS. Data is read the clock cycle s<LAYER_NUM>_axis_tvalid and s<LAYER_NUM>_axis_tready are both High.<br>*m* is C_DATA_WIDTH for the following bit definitions.<br>Data format for Layer 0 (2 Channels):<br>• Bits (n-1)–3*m: RESERVED(3)<br>• Bits (3*m-1)–2*m: Alpha Channel<br>• Bits (2*m-1)–m: Data Channel 1<br>• Bits (m-1)–0: Data Channel 0<br>Data format for Layer 0 (3 Channels):<br>• Bits (n-1)–4*m: RESERVED(3)<br>• Bits (4*m-1)–3*m: Alpha Channel<br>• Bits (3*m-1)–2*m: Data Channel 2<br>• Bits (2*m-1)–m: Data Channel 1<br>• Bits (m-1)–0: Data Channel 0<br>Data format for Layers 1–7 is the same for Layer 0. |
| s<*LAYER_NUM*>_axis_ tkeep | I | [n/8-1: 0](1) | AXI4- STREAM WRITE KEEP<br>Indicates valid bytes on AXI4-Stream data.<br>• 1 = Write data byte is valid.<br>• 0 = Write data byte is not valid. |
| s<*LAYER_NUM*>_axis_ tvalid | I | 1 | AXI4- STREAM VALID IN<br>Indicates AXI4-Stream data bus, s<*LAYER_NUM*>_axis_tdata, is valid.<br>• 1 = Write data is valid.<br>• 0 = Write data is not valid. |

*Table 2-1:* **Common Port Descriptions *(Cont'd)***

| Port Name | Dir | Width | Description |
|---|---|---|---|
| s<*LAYER_NUM*>_axis_ tready | O | 1 | AXI4- STREAM READY<br>Indicates AXI4-Stream target is ready to receive stream data.<br>• 1 = Ready to receive data.<br>• 0 = Not ready to receive data. |
| s<*LAYER_NUM*>_axis_ tlast | I | 1 | AXI4-STREAM LAST<br>Indicates last data beat per video line of AXI4-Stream data.<br>• 1 = Last data beat of video line.<br>• 0 = Not last data beat. |
| **Master AXI4-Stream Interface** | | | |
| m_axis_tdata | O | [n -1: 0] (2) | AXI4- STREAM DATA OUT<br>Output AXI4-Stream data. Data format is the same as the s0_axis_tdata format except the m_axis_tdata bus has no alpha channel. |
| m_axis_tkeep | O | [n/8-1: 0] (2) | AXI4- STREAM WRITE KEEP<br>Indicates valid bytes on AXI4-Stream data.<br>• 1 = Write data byte is valid.<br>• 0 = Write data byte is not valid. |
| m_axis_ tvalid | O | 1 | AXI4- STREAM VALID OUT<br>Indicates AXI4-Stream data bus, m_axis_tdata, is valid.<br>• 1 = Write data is valid.<br>• 0 = Write data is not valid. |
| m_axis_ tready | I | 1 | AXI4- STREAM READY<br>Indicates AXI4-Stream target is ready to receive stream data.<br>• 1 = Ready to receive data.<br>• 0 = Not ready to receive data. |
| m_axis_ tlast | O | 1 | AXI4-STREAM LAST<br>Indicates last data beat per video line of AXI4-Stream data.<br>• 1 = Last data beat of video line.<br>• 0 = Not last data beat. |
| **Output Xilinx Streaming Video Interface** | | | |
| video_clk_out | O | 1 | OUTPUT VIDEO CLOCK<br>This port is only present on the EDK pCore Interface. |
| vblank_out | O | 1 | OUTPUT VERTICAL BLANKING PERIOD SIGNAL<br>Delayed vblank_in. |

*Table 2-1:* **Common Port Descriptions *(Cont'd)***

| Port Name | Dir | Width | Description |
|---|---|---|---|
| vsync_out | O | 1 | OUTPUT VERTICAL SYNCHRONIZATION SIGNAL<br>Delayed vsync_in. |
| hblank_out | O | 1 | OUTPUT HORIZONTAL SYNCHRONIZATION SIGNAL<br>Delayed hblank_in. |
| hsync_out | O | 1 | OUTPUT HORIZONTAL BLANKING PERIOD SIGNAL<br>Delayed hsync_in. |
| active_video_out | O | 1 | OUTPUT ACTIVE VIDEO<br>Delayed active_video_in. |
| active_chroma_out | O | 1 | OUTPUT ACTIVE CHROMA<br>Delayed active_chroma_in. |
| video_data_out | O | [C_NUM_DATA_CHANNELS*C_DATA_WIDTH-1:0] | OUTPUT VIDEO DATA<br>The resultant output data of the OSD after alpha blending all layers. Data format is the same as the vfbc_rd_data format for layer 0 for all data channels. The video_data_out port has no alpha channel. |

1. The data width, n of the s<LAYER_NUM>_axis_tdata bus is calculated as the next power of 2 greater than the data channel width multiplied by the number of data channels including the alpha channel, or (C_NUM_DATA_CHANNELS+C_ALPHA_CHANNEL_EN)*C_DATA_WIDTH.
2. The data width, n, of the m_axis_tdata bus is calculated as the next power of 2 greater than the data channel width multiplied by the number of data channels excluding the alpha channel, or C_NUM_DATA_CHANNELS*C_DATA_WIDTH.
3. All reserved input pins must be driven by '0'.
4. LAYER_NUM in the Slave AXI4-Stream interfaces indicates the layer number for that input. For example, if layer 3 is configured for AXI4-Stream Input, then the ports for this input are s3_axis_tdata, s3_axis_tkeep, s3_axis_tvalid, s3_axis_tready, and s3_axis_tlast.

## EDK pCore Interface

The pCore interface creates a core that can be easily added to an EDK Project as a hardware peripheral. Table 2-2 describes the I/O signals associated with the OSD pCore.

*Table 2-2:* **AXI4-Lite pCore I/O Signals**

| Port Name | Dir | Width | Description |
|---|---|---|---|
| **AXI Global System Signals(1)** | | | |
| S_AXI_ARESETN | I | 1 | AXI Reset, active Low |
| IP2INTC_Irpt | O | 1 | Interrupt request output |
| **AXI Write Address Channel Signals(1)** | | | |
| 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. |

*Table 2-2:* **AXI4-Lite pCore I/O Signals**

| Port Name | Dir | Width | Description |
|---|---|---|---|
| S_AXI_AWVALID | I | 1 | AXI4-Lite Write Address Channel Write Address Valid. This signal indicates that valid write address is available.<br>1 = Write address is valid.<br>0 = Write address is not valid. |
| S_AXI_AWREADY | O | 1 | AXI4-Lite Write Address Channel Write Address Ready. Indicates core is ready to accept the write address.<br>1 = Ready to accept address.<br>0 = Not ready to accept address. |
| **AXI Write Data Channel Signals(1)** | | | |
| 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.<br>1 = Write data/strobes are valid.<br>0 = Write data/strobes are not valid. |
| S_AXI_WREADY | O | 1 | AXI4-Lite Write Data Channel Write Data Ready. Indicates core is ready to accept the write data.<br>1 = Ready to accept data.<br>0 = Not ready to accept data. |
| **AXI Write Response Channel Signals(1)** | | | |
| S_AXI_BRESP(2) | O | [1:0] | AXI4-Lite Write Response Channel. Indicates results of the write transfer.<br>00b = OKAY - Normal access has been successful.<br>01b = EXOKAY - Not supported.<br>10b = SLVERR - Error.<br>11b = DECERR - Not supported. |
| S_AXI_BVALID | O | 1 | AXI4-Lite Write Response Channel Response Valid. Indicates response is valid.<br>1 = Response is valid.<br>0 = Response is not valid. |
| S_AXI_BREADY | I | 1 | AXI4-Lite Write Response Channel Ready. Indicates Master is ready to receive response.<br>1 = Ready to receive response.<br>0 = Not ready to receive response. |

*Table 2-2:* **AXI4-Lite pCore I/O Signals**

| Port Name | Dir | Width | Description |
|---|---|---|---|
| **AXI Read Address Channel Signals[1]** | | | |
| 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.<br>1 = Read address is valid.<br>0 = Read address is not valid. |
| S_AXI_ARREADY | O | 1 | AXI4-Lite Read Address Channel Read Address Ready. Indicates core is ready to accept the read address.<br>1 = Ready to accept address.<br>0 = Not ready to accept address. |
| **AXI Read Data Channel Signals[1]** | | | |
| 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.<br>00b = OKAY - Normal access has been successful.<br>01b = EXOKAY - Not supported.<br>10b = SLVERR - Error.<br>11b = DECERR - Not supported. |
| 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.<br>1 = Read data is valid.<br>0 = Read data is not valid. |
| S_AXI_RREADY | I | 1 | AXI4-Lite Read Data Channel Read Data Ready. Indicates master is ready to accept the read data.<br>1 = Ready to accept data.<br>0 = Not ready to accept data. |

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.

## General Purpose Processor Interface

The GPP Interface consists of the Common I/O signals listed in Table 2-1 and the GPP ports detailed in Table 2-3. The GPP ports in Table 2-3 correspond to the AXI4-Lite registers on the EDK pCore found in Table 2-4.

The directly exposed Dynamic Configuration Interface signals allow the user to wrap these signals with a user-defined bus interface targeting any arbitrary processor. It is recommended to disable the `control[2]` (Register Update Enable) signal of the control bus before updating the other Dynamic Configuration Interface signals. After the Dynamic Configuration Interface signals are ready to be updated in the core, the `control[2]`

signal should be enabled. Values are written into the core on the falling edge of the `frame_sync` input.

*Table 2-3:* **General Purpose Processor Port Descriptions**

| Port Name | Dir | Width | Description |
|---|---|---|---|
| sclr | I | 1 | SYNCHRONOUS CLEAR/RESET<br>System synchronous reset (active High). Asserting `sclr` synchronously with `clk` resets the Xilinx On- Screen Display internal state machines. sclr has priority over `ce`. |
| control | I | [31:0] | OSD CONTROL REGISTER<br>Bits 31–6: RESERVED(1)<br>Bit 5: Input vertical blank polarity<br>Bit 4: Input horizontal blank polarity<br>Bit 3: RESERVED(1)<br>Bit 2: OSD Register Update Enable<br>Bit 1: RESERVED(1)<br>Bit 0: OSD Enable |
| bgcolor0 | I | [C_DATA_WIDTH-1:0] | Background Color Channel 0<br>Background color component for channel 0. If the number of channels is set to two, then this must be the luminance (Y) value. |
| bgcolor1 | I | [C_DATA_WIDTH-1:0] | Background Color Channel 1<br>Background color component for channel 1. If the number of channels is set to two, then this must be the chrominance (Cb) value. |
| bgcolor2 | | [C_DATA_WIDTH-1:0] | Background Color Channel 2<br>Background color component for channel 2. If the number of channels is set to two, then this must be the chrominance (Cr) value. |
| layer_enable | I | [C_NUM_LAYERS-1 : 0] | LAYER ENABLES<br>Each bit controls one layer. Bits 7–0 correspond to layers 7–0 respectively.<br>0: Layer does not show on screen<br>1: Layer is enabled and shows on screen |

*Table 2-3:* **General Purpose Processor Port Descriptions *(Cont'd)***

| Port Name | Dir | Width | Description |
|---|---|---|---|
| layer_priority | I | [C_NUM_LAYERS*32-1 : 0] | LAYER PRIORITIES<br>Controls the Z-plane order of each layer.<br>Bits 31–3: RESERVED(1)<br>Bits 2–0: Layer 0 priority<br>Bits 255–32 are repeated for layers 1–7.<br>A layer with a lower priority than another layer will be displayed beneath. 0 is the lowest priority; 7 is the highest priority. Each layer must have a unique valid priority number. The maximum valid priority number is C_NUM_LAYERS-1. If two or more layers have the same priority, then they will not be displayed correctly or at all. |
| layer_alpha | I | [C_NUM_LAYERS*32-1 : 0] | LAYER ALPHA CONTROL<br>Global Alpha values and Global Alpha Enables for each layer.<br>Bit 31: Layer 0 Global Alpha enable<br>When High, the pixel alpha from the `vfbc_rd_data` is ignored and the global alpha value from bits 7–0 is used for every pixel in this layer.<br>Bits 30–C_DATA_WIDTH: RESERVED(1)<br>Bits (C_DATA_WIDTH-1)–0: Global Alpha value<br>Bits 255–32 are repeated for layers 1–7.<br>***Note:*** Useful for YUV or RGB formats that do not have a corresponding pixel alpha. |
| layer_x_pos | I | [C_NUM_LAYERS*32-1 : 0] | LAYER STARTING X POSITION<br>Horizontal start pixel of origin of each layer. Origin of screen is located at (0,0).<br>Bits 31–12: RESERVED(1)<br>Bits 11–0: Layer 0 horizontal start pixel<br>This is the pixel of the layer origin (upper-left corner).<br>Bits 255–32 are repeated for layers 1–7<br>If the OSD is configured for 2-channels, then the layer starting x position should be set to an even pixel boundary to avoid artifacts. |

*Table 2-3:* **General Purpose Processor Port Descriptions *(Cont'd)***

| Port Name | Dir | Width | Description |
|---|---|---|---|
| layer_y_pos | I | [C_NUM_LAYERS*32-1 : 0] | LAYER STARTING Y POSITION<br>Vertical start line of origin of each layer. Origin of screen is located at (0,0).<br>Bits 31–12: RESERVED(1)<br>Bits 11–0: Layer 0 vertical start line<br>This is the line of the layer origin (upper-left corner).<br>Bits 255–32 are repeated for layers 1–7. |
| layer_x_size | I | [C_NUM_LAYERS*32-1 : 0] | LAYER X SIZE<br>Horizontal Size of Each Layer.<br>Bits 31–12: RESERVED(1)<br>Bits 11–0: Layer 0 horizontal width in number of pixels<br>Bits 255–32 are repeated for layers 1–7.<br>If the OSD is configured for 2-channels, then the layer x size should be set to an even pixel value to avoid artifacts. |
| layer_y_size | I | [C_NUM_LAYERS*32-1 : 0] | LAYER Y SIZE<br>Vertical Size of Each Layer.<br>Bits 31–12: RESERVED(1)<br>Bits 11–0: Layer 0 vertical height in number of lines<br>Bits 255–32 are repeated for layers 1–7. |
| screen_x | I | [11:0] | Output Screen X Size<br>Horizontal Width of OSD Output. |
| screen_y | I | [11:0] | Output Screen Y Size<br>Vertical Height of OSD Output. |
| intr_status | O | [31:0] | INTERRUPT STATUS<br>Active High edge interrupt status register. |
| version | O | [31:0] | CORE VERSION<br>Bits 31-28: Major version as a single 4-bit hexadecimal value<br>Bits 27-20: Minor version as two separate 4-bit hexadecimal values (00 – FF)<br>Bits 19:16: Revision letter as a hexadecimal character from 'a' – 'f'; mapping is as follows: 0XA-'>a', 0xB->'b', 0xC->'c', 0xD->'d', etc<br>Bits 15-12: CORE Generator™ tool Patch Revision<br>Bits 11-0: RESERVED(1) |

*Table 2-3:* **General Purpose Processor Port Descriptions *(Cont'd)***

| Port Name | Dir | Width | Description |
|---|---|---|---|
| **Graphics Controller Interface** | | | |
| gc_active_bank_addr | I | [31:0] | Active Bank Address<br>Sets the Active Memory Bank for each RAM in each Graphics Controller (GC). For all bits: '0' selects RAM 0, '1' selects RAM 1.<br>Bits 31-24: Active Font RAM for GC 7-0<br>Bits 23-16: Active Text RAM for GC 7-0<br>Bits 15-8: Active Color Bank for GC 7-0<br>Bits 7-0: Active Instruction Bank for GC 7-0 |
| gc_write_bank_addr | I | [5:0] | Write Bank Address<br>Controls which memory bank to write data.<br>Bits 5–3: The Graphics Controller Layer Number<br>Selects which Graphics Controller to write to.<br>Bits 2–0:<br>000: Write data into Instruction RAM 0<br>001: Write data into Instruction RAM 1<br>010: Write data into Color RAM 0<br>011: Write data into Color RAM 1<br>100: Write data into Text RAM 0<br>101: Write data into Text RAM 1<br>110: Write data into Font RAM 0<br>111: Write data into Font RAM 1 |
| gc_write_bank_we | I | 1 | Write Bank Address Write Enable |
| gc_data | I | [31:0] | GRAPHICS CONTROLLER DATA<br>The data from the host to be written to internal Graphics Controller memory. |
| gc_data_we | I | 1 | GRAPHICS CONTROLLER write enable<br>Data is written to the currently selected memory bank when the gc_data_we is active. The memory address is automatically incremented after each write. Active High. |

1. All reserved input pins must be driven by '0'.

See the AMBA AXI4 Interface Protocol web site (http://www.xilinx.com/ipcenter/axi4.htm) for more information on the AXI4 and AXI4-Lite ports and operation.

# I/O Interface and Timing

This section describes the signals and timing of the different interfaces of the Xilinx Video On-Screen Display.

## Input AXI4-Stream Slave Interface(s)

The Xilinx Video On-Screen Display can be configured to have up to eight input AXI4-stream slave interfaces. These interfaces include and require the TDATA, TKEEP, TVALID, TREADY and TLAST AXI4-Stream signals. The `s<LAYER_NUM>_axis_tkeep` (TKEEP) bus must be asserted to all ones for every valid `s<LAYER_NUM>_axis_tdata` (TDATA) transfer. The `s<LAYER_NUM>_axis_tlast` (TLAST) must be asserted High during the last TDATA transaction of each video line. The `s<LAYER_NUM>_axis_tdata` (TDATA) width must be a power of 2, with valid widths of 8, 16, 32, or 64. Unused bits should be driven by zero.

Figure 2-1 shows example AXI4-Stream transactions for two video frames that are 5 pixels by 2 lines.

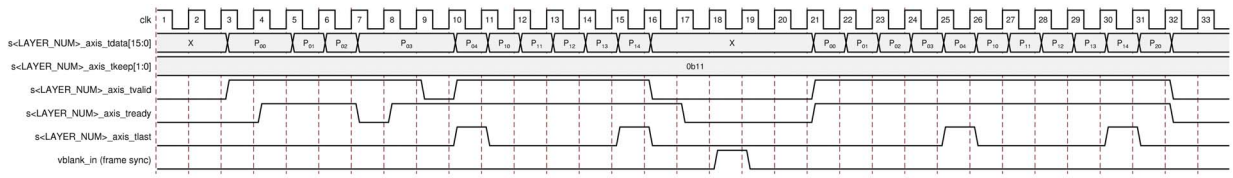



*Figure 2-1:* **Input AXI4-Stream Timing**

Figure 2-1 shows that the `s<LAYER_NUM>_axis_tlast` port is asserted High during the last pixel transfer of each line, denoted by $P_{04}$ and $P_{14}$s.

In addition, the final pixel transfer of the last video line must complete before the start of the vertical blank period. The OSD uses the `vblank_in` port (for both output modes of XSVI output and AXI4-Stream) to detect the frame sync. The rising edge of an active-High polarity `vblank_in` or the falling edge of an active Low polarity `vblank_in` will trigger an internal frame sync event. The OSD control register is used to set the expected polarity of the `vblank_in` signal. This implies that the AXI4-Stream throughput must be high enough to ensure that the frame completes before the start of vblank or the frame sync pulse.

## Output AXI4-Stream Master Interface

The output interface of the Xilinx Video On-Screen Display can be configured to be a AXI4-Stream interface. This interface includes and requires the TDATA, TKEEP, TVALID, TREADY and TLAST AXI4-Stream signals. The `m_axis_tkeep` (TKEEP) bus will be driven to all ones for every valid `m_axis_tdata` (TDATA) transfer. The `m_axis_tlast` (TLAST) will be driven High during the last TDATA transaction of each video line. The `m_axis_tdata` (TDATA) width must be a power of 2, with valid widths of 8, 16, 32, or 64. Unused bits will be driven by zero.

Figure 2-2 shows example AXI4-Stream transactions for 2 video frames of size 5 pixels by 2 lines.

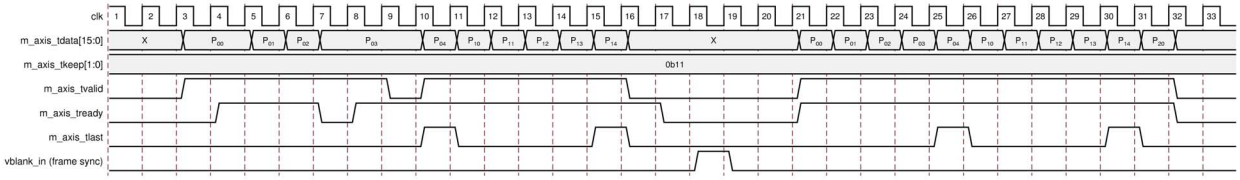



*Figure 2-2:* **Output AXI4-Stream Timing**

Figure 2-2 shows that the `m_axis_tlast` port is driven High during the last pixel transfer of each line, denoted by $P_{04}$ and $P_{14}$.

The final pixel transfer of the last video line must complete before the start of the vertical blank period. The OSD uses the `vblank_in` port (for both output modes of XSVI output and AXI4-Stream) to detect the frame sync. The rising edge of an active High polarity `vblank_in` or the falling edge of an active Low polarity vblank_in will trigger an internal frame sync event. The OSD control register is used to set the expected polarity of the `vblank_in` signal. This implies that the AXI4-Stream throughput must be high enough to ensure that the frame completes before the start of vblank or the frame sync pulse.

## Xilinx Streaming Video Interface (XSVI)

The output interface of the Xilinx Video On-Screen Display can be configured to be a Xilinx Streaming Video Interface (XSVI). In this mode, valid data is presented on the `video_data_out` output bus in raster order. The `video_data_out` bus is aligned to six video timing signal outputs. These video timing signals outputs are `vblank_out` (for denoting the vertical blanking period), `vsync_out` (for denoting the vertical sync), `hblank_out` (for denoting the horizontal blanking period), `hsync_out` (for denoting the horizontal sync), `active_video_out` (for denoting those clock cycles that contain valid data on the `video_data_out` bus) and active_chroma_out (for denoting those video lines that contain valid chroma).

Figure 2-3 shows an example XSVI output waveform. All video timing signals are shown as active High polarity. The output video frame in this example is configured for four active video lines, three lines of vertical blanking and one line of vertical sync.

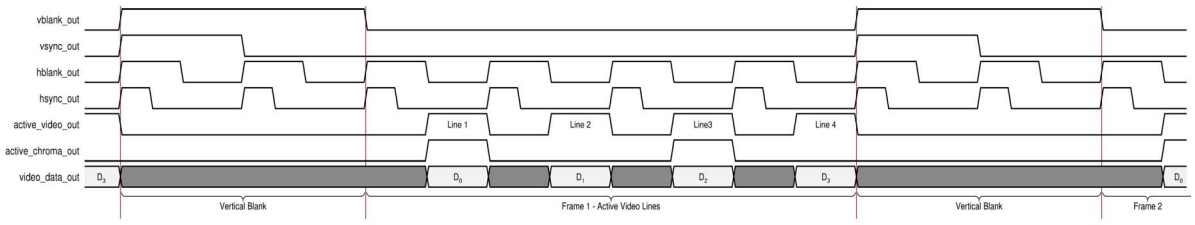



*Figure 2-3:* **XSVI Output Timing**

The video timing signal outputs are not generated internally by the OSD. Instead there are six similar video timing signal inputs. These inputs are delayed and present on the XSVI output. Valid data on the `video_data_out` output bus is aligned to the video timing signal outputs.

Each video timing signal input may be active High or active Low. Only the vertical and horizontal blank signals are used by the OSD. The vertical and horizontal sync signals and

the active video signal are optional and required only if the hardware connected to the XSVI output of the OSD expects these signals.

The polarity of the horizontal and vertical blank input signals must be known and programmed in the OSD Control Register.

## Graphics Controller Host Interface

Data is written into the internal memory of each graphics controller via a single host interface. The Xilinx Video On-Screen Display provides five input ports for controlling and loading this data. The `gc_write_bank_addr` port selects which internal memory bank (RAM) of which graphics controller to be written. The `gc_write_bank_addr` is captured the first clock cycle that the `gc_write_bank_we` (bank write enable) port is asserted High. The `gc_data` port should be driven with the data to be written to internal memory during the first clock cycle the `gc_data_we` (data write enable) port is asserted High. The `gc_active_bank_addr` selects the active memory bank for instruction, font, text and color memories for each graphics controller.

Figure 2-4 shows an example graphics controller memory load operation for two memories.

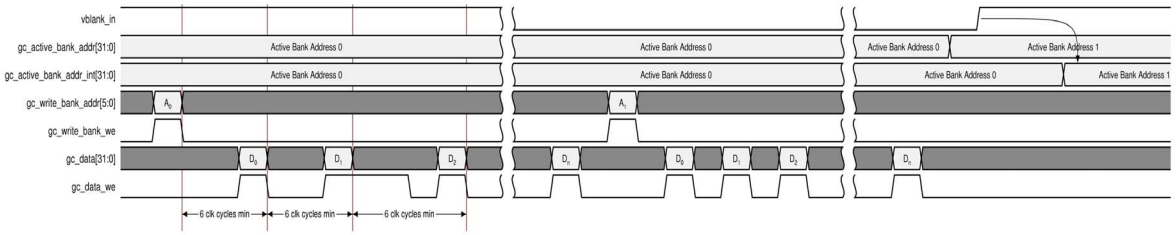



*Figure 2-4:* **Graphics Controller Host Interface Timing**

To write into the internal memory of a graphics controller, the host processor must:

1. Write the bank address to the `gc_write_bank_addr` register. This selects the target memory of the target graphics controller.
2. Write all data in single cycle writes until all data is written. The graphics controller will automatically increment its internal address pointer after each write. The graphics controller will also set its corresponding address overflow interrupt status bit if the host tries to write beyond the depth of the currently selected memory.

For the new data to be used by the graphics controller, the host processor must set the memory active in the `gc_active_bank_addr` register after all data is written. The graphics controller will use the new setting after the start of the next vertical blanking interval period defined by the `vblank_in` port. The host processor should avoid writing to memory that is currently set active in the `gc_active_bank_addr` register.

There must be at least six clock cycles between each write to the `gc_write_bank_addr` or `gc_data` registers. The `gc_data_we` input can be asserted High for more than one clock cycle. The `gc_data` bus is read only during the first clock cycle that the `gc_data_we` is High. The `gc_write_bank_we` and `gc_write_bank_addr` behave similarly. The `gc_write_bank_addr` is only read during the first cycle that the `gc_write_bank_we` is asserted High.

The OSD contains and keeps track of multiple address pointers internally and increments after each `gc_data_we`. The internal address pointer(s) are reset when the `gc_write_bank_we` is asserted High. There are three internal address pointers, one for the instruction RAM, one for the color RAM and one for font and text RAMs. Depending

on the `gc_write_bank_addr` value(00=instruction, 01=color, 10=text, 11=font), one of the three pointers is reset.

## Interrupts

The Xilinx Video On-Screen Display provides a 32-bit output bus, `intr_status`[31:0], for host processor interrupt status when configured for the General Purpose Processor (GPP) interface. All interrupt status bits can trigger an interrupt on the active High edge. Status bits are set High when the internal event occurs and are cleared ether at the start or at the end of the vertical blanking interval period defined by the `vblank_in` port.

Interrupt status bits 31-3 are cleared at the start of the vertical blanking interval period. These bits include the graphics controller address overflow, the graphics controller instruction error, the output FIFO overflow error, the input FIFOs underflow error and the vertical blanking interval end interrupt status bits.

Interrupt status bits 2-0 are cleared at the end of the vertical blanking interval period. These bits include the vertical blanking interval period start, frame error and frame done interrupt status bits.

The interrupt status output bus can easily be integrated with an external interrupt controller that has independent interrupt enable/mask, interrupt clear and interrupt status registers and that allows for interrupt aggregation to the system processor. An example system showing the OSD and other processor peripherals connected to an interrupt controller is depicted in Figure 2-5.

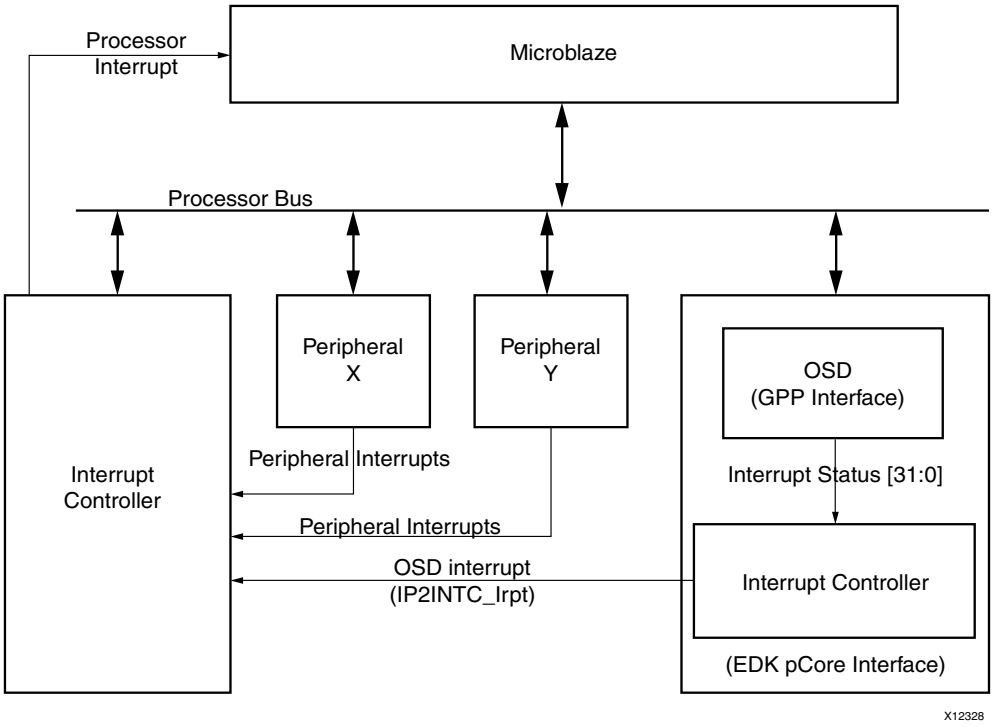



*Figure 2-5:* **Interrupt Controller Processor Peripherals**

The Xilinx Video On-Screen Display, when configured for the EDK pCore Interface, automatically contains an internal interrupt controller for enabling/masking and clearing each interrupt. The 1-bit output port, `IP2INTC_Irpt`, is the interrupt output in this mode. The OSD internal interrupt controller is described in DS516, *LogiCORE IP Interrupt Control Data Sheet*.

## General Purpose Processor Interface

The General Purpose Processor Interface is a dynamic register interface and exposes all control and status registers as ports. These ports can easily be connected to any processor with a user-defined bus interface via a Register File and minimal logic. The interrupt status register is included in this interface.

All input ports are double-buffered and captured internally at the start of the vertical blanking interval period defined by the `vblank_in` port. In addition, the register update enable (bit 2 of the OSD Control Register) can disable updates to the internally buffered registers. This allows the host processor to update OSD control registers during multiple video frames. The register update enable bit is not double-buffered.

The General Purpose Processor ports for this core are defined in Table 2-3, page 45.

## EDK pCore AXI4-Lite Interface

The Xilinx Video On-Screen Display, when configured as an EDK pCore, uses the AXI4-Lite Interface to interface to a microprocessor. Refer to the AMBA AXI4 Interface Protocol website (http://www.xilinx.com/ipcenter/axi4.htm) for more information on the AXI4 and AXI4-Lite interface signals.

When the developer selects the EDK pCore interface, Xilinx CORE Generator™ creates a pCore and all support files 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 and a complete device driver to enable rapid application development.

Xilinx CORE Generator will place all EDK pCore source files in the "pcores" subdirectory located in the core output directory. The core output directory is given the same name as the component. For example, if the component name is set to "v_osd_v3_0_u0," then the EDK pCore source files will be located in the following directory:

```
<coregen project directory>/v_osd_v3_0_u0/pcores/
axi_osd_v3_00_a
```

The pCore should be copied to the user's `<EDK_Project>/pcores` directory or to a user pCores repository.

# Register Space

This section contains details about the OSD registers.

## pCore Address Map

All registers default to 0x00000000 on power-up or software reset.

*Table 2-4:* **EDK pCore Address Map**

| Address Offset | Name | Read/ Write | Description |
|---|---|---|---|
| 0x0000 | OSD Control | R/W | General control register |
| 0x0004 | Reserved | – | |
| 0x0008 | Reserved | – | |
| 0x000C | Reserved | – | |
| 0x0010 | OSD Screen Size | R/W | X and Y Size of the screen |
| 0x0014 | OSD Background Color 0 | R/W | Background Color Channel 0 |
| 0x0018 | OSD Background Color 1 | R/W | Background Color Channel 1 |
| 0x001C | OSD Background Color 2 | R/W | Background Color Channel 2 |
| 0x0020 | OSD Layer 0 Control | R/W | Video Layer Enable, Priority, Alpha |
| 0x0024 | OSD Layer 0 Position | R/W | Video Layer Position |
| 0x0028 | OSD Layer 0 Size | R/W | Video Layer Size |
| 0x002c | Reserved | – | |
| 0x0030 | OSD Layer 1 Control | R/W | Video Layer Enable, Priority, Alpha |
| 0x0034 | OSD Layer 1 Position | R/W | Video Layer Position |
| 0x0038 | OSD Layer 1 Size | R/W | Video Layer Size |
| 0x003c | Reserved | – | |
| 0x0040 | OSD Layer 2 Control | R/W | Video Layer Enable, Priority, Alpha |

*Table 2-4:* **EDK pCore Address Map *(Cont'd)***

| Address Offset | Name | Read/ Write | Description |
|---|---|---|---|
| 0x0044 | OSD Layer 2 Position | R/W | Video Layer Position |
| 0x0048 | OSD Layer 2 Size | R/W | Video Layer Size |
| 0x004c | Reserved | – | |
| 0x0050 | OSD Layer 3 Control | R/W | Video Layer Enable, Priority, Alpha |
| 0x0054 | OSD Layer 3 Position | R/W | Video Layer Position |
| 0x0058 | OSD Layer 3 Size | R/W | Video Layer Size |
| 0x005c | Reserved | – | |
| 0x0060 | OSD Layer 4 Control | R/W | Video Layer Enable, Priority, Alpha |
| 0x0064 | OSD Layer 4 Position | R/W | Video Layer Position |
| 0x0068 | OSD Layer 4 Size | R/W | Video Layer Size |
| 0x006c | Reserved | – | |
| 0x0070 | OSD Layer 5 Control | R/W | Video Layer Enable, Priority, Alpha |
| 0x0074 | OSD Layer 5 Position | R/W | Video Layer Position |
| 0x0078 | OSD Layer 5 Size | R/W | Video Layer Size |
| 0x007c | Reserved | – | |
| 0x0080 | OSD Layer 6 Control | R/W | Video Layer Enable, Priority, Alpha |
| 0x0084 | OSD Layer 6 Position | R/W | Video Layer Position |
| 0x0088 | OSD Layer 6 Size | R/W | Video Layer Size |
| 0x008c | Reserved | – | |
| 0x0090 | OSD Layer 7 Control | R/W | Video Layer Enable, Priority, Alpha |
| 0x0094 | OSD Layer 7 Position | R/W | Video Layer Position |
| 0x0098 | OSD Layer 7 Size | R/W | Video Layer Size |
| 0x009c | Reserved | – | |
| 0x00a0 | OSD GC Write Bank Address | R/W | Graphics Controller Write Bank Address.<br>Used for all Instantiated Graphics Controllers |
| 0x00a4 | OSD GC Active Bank Address | R/W | Graphics Controller Active Bank Addresses. Selected after next vblank.<br>Used for all Instantiated Graphics Controllers |

*Table 2-4:* **EDK pCore Address Map *(Cont'd)***

| Address Offset | Name | Read/ Write | Description |
|---|---|---|---|
| 0x00a8 | OSD GC Data | R/W | Graphics Controller Data Register<br>Used to write instructions, Character Map, ASCII text strings and color.<br>Used for all Instantiated Graphics Controllers. |
| 0x00ac - 0x00ec | Reserved | – | |
| 0x00f0 | Version Register | R | Core Hardware Version |
| 0x0100 | OSD Software Reset | R/W | |
| 0x021c | OSD GIER | R/W | OSD Global Interrupt Enable Register |
| 0x0220 | OSD ISR - Interrupt Status/Clear | R/W | General readable status register for polling error |
| 0x0228 | OSD IER - Interrupt Enable | R/W | General read/write interrupt Enable Register |

*NOTE: The registers in both the EDK pCore Interface and General Purpose Processor Interface are little endian.*

*Table 2-5:* **OSD Control Register (Address Offset 0x0000)**

| 0x0000 | OSD Control | R/W |
|---|---|---|
| Name | Bits | Description |
| Reserved | 31:6 | Reserved |
| V Blank Polarity | 5 | INPUT Vertical Blank Polarity<br>1: The polarity of the v blank input is active High.<br>0: The polarity of the v blank input is active Low. |
| H Blank Polarity | 4 | INPUT Horizontal Blank Polarity<br>1: The polarity of the h blank input is active High.<br>0: The polarity of the h blank input is active Low. |
| Reserved | 3 | Reserved |
| OSD Register Update Enable | 2 | OSD Register Update Enable<br>Setting this bit to 1 will cause the OSD to re-read all register values after the next frame sync or vertical blank.<br>Setting this bit to 0 will cause the OSD to use its internally buffered register values.<br>This Register update enable is used only for address 0x10 - 0x9c. Not used for Graphics Controller Registers. |
| Reserved | 1 | Reserved |
| OSD Enable | 0 | Enable/Start the OSD<br>This will cause the OSD to start reading from external memory and writing output after the next Vsync. |

*Table 2-6:* **OSD Screen Size Register (Address Offset 0x0010)**

| 0x0010 | OSD Screen Size | R/W |
|---|---|---|
| Name | Bits | Description |
| Reserved | 31:28 | |
| Y size | 27:16 | Vertical Height of OSD Output |
| Reserved | 15:12 | |
| X size | 11:0 | Horizontal Width of OSD Output |

*Table 2-7:* **OSD Background Color Register (Address Offset 0x0014)**

| 0x0014 | | R/W |
|---|---|---|
| Name | Bits | Description |
| Reserved | 31:C_DATA_WIDTH | |
| Background Color 0 | C_DATA_WIDTH-1:0 | Background Color component of channel 0<br>Typically, Y (luma) or Green |

*Table 2-8:* **OSD Background Color Register (Address Offset 0x0018)**

| 0x0018 | OSD Background Color Channel 1 | R/W |
|---|---|---|
| Name | Bits | Description |
| Reserved | 31:C_DATA_WIDTH | |
| Background Color 1 | C_DATA_WIDTH-1:0 | Background Color component of channel 1<br>Typically, U (Cb) or Blue |

*Table 2-9:* **OSD Background Color Register (Address Offset 0x001C)**

| 0x001c | OSD Background Color Channel 2 | R/W |
|---|---|---|
| Name | Bits | Description |
| Reserved | 31:C_DATA_WIDTH | |
| Background Color 2 | C_DATA_WIDTH-1:0 | Background Color component of channel 2<br>Typically, V (Cr) or Red |

*Table 2-10:* **OSD Layer 0 Control Register (Address Offset 0x0020)**

| 0x0020 | OSD Layer 0 Control | R/W |
|---|---|---|
| Name | Bits | Description |
| Reserved | 31:16+C_DATA_WIDTH | |
| Alpha | 16+(C_DATA_WIDTH-1):16 | Layer 0 Global Alpha Value<br>0 = Layer 100% transparent<br>255 = Layer 0% transparent (100% opaque) |
| Reserved | 15:11 | |
| Priority | 10:8 | Layer 0 Priority<br>0 = Lowest<br>1 = Higher<br>..<br>7 = Highest |
| Reserved | 7:2 | |
| Layer0_Galpha_en | 1 | Layer 0 Global Alpha Enable |
| Layer0_en | 0 | Layer 0 Enable |

*Table 2-11:* **OSD Layer 0 Position Register (Address Offset 0x0024)**

| 0x0024 | OSD Layer 0 Position | R/W |
|---|---|---|
| Name | Bits | Description |
| Reserved | 31:28 | Reserved |
| Y position | 27:16 | Vertical start line of origin of layer. Origin of screen is located at (0,0). |
| Reserved | 15:12 | |
| X position | 11:0 | Horizontal start pixel of origin of layer. Origin of screen is located at (0,0). |

*Table 2-12:* **OSD Layer 0 Size Register (Address Offset 0x0028)**

| 0x0028 | OSD Layer 0 Size | R/W |
|---|---|---|
| Name | Bits | Description |
| Reserved | 31:28 | |
| Y size | 27:16 | Vertical Size of Layer |
| Reserved | 15:12 | |
| X size | 11:0 | Horizontal Size of Layer |

***Note:*** 0x20 – 0x2C are repeated for Layers 1 through 7 at addresses 0x30-0x9c.

*Table 2-13:* **OSD GC Write Bank Address Register (Address Offset 0x00A0)**

| 0x00A0 | OSD GC Write Bank Address | R/W |
|---|---|---|
| Name | Bits | Description |
| Reserved | 31:11 | |
| GC Number | 10:8 | Graphics Controller Number<br>The Graphics Controller Layer Number. If a layer is configured for a graphics controller, then setting the layer number here will allow writing data to that graphics controller. |
| Reserved | 7:3 | |
| GC_Write_Bank_ Addr | 2:0 | **OSD GC Bank Write Address**<br>Controls which memory bank to write data.<br>000: Write data into Instruction RAM 0<br>001: Write data into Instruction RAM 1<br>010: Write data into Color RAM 0<br>011: Write data into Color RAM 1<br>100: Write data into Text RAM 0<br>101: Write data into Text RAM 1<br>110: Write data into Font RAM 0<br>111: Write data into Font RAM 1 |

*Table 2-14:* **OSD GC Active Bank Address Register (Address Offset 0x00A4)**

| 0x00A4 | OSD GC Active Bank Address | R/W |
|---|---|---|
| Name | Bits | Description |
| GC_Char_ActBank | 31:24 | Sets the Active CharacterMap/Font Bank.<br>Bit 31 = Active Font RAM Bank for GC 7<br>Bit 30 = Active Font RAM Bank for GC 6<br>Bit 29 = Active Font RAM Bank for GC 5<br>Bit 28 = Active Font RAM Bank for GC 4<br>Bit 27 = Active Font RAM Bank for GC 3<br>Bit 26 = Active Font RAM Bank for GC 2<br>Bit 25 = Active Font RAM Bank for GC 1<br>Bit 24 = Active Font RAM Bank for GC 0 |
| GC_Text_ActBank | 23:16 | Sets the active Text Bank.<br>Bit 23 = Active Text RAM Bank for GC 7<br>Bit 22 = Active Text RAM Bank for GC 6<br>Bit 21 = Active Text RAM Bank for GC 5<br>Bit 20 = Active Text RAM Bank for GC 4<br>Bit 19 = Active Text RAM Bank for GC 3<br>Bit 18 = Active Text RAM Bank for GC 2<br>Bit 17 = Active Text RAM Bank for GC 1<br>Bit 16 = Active Text RAM Bank for GC 0 |
| GC_Col_ActBank | 15:8 | Sets the active Color Table Bank.<br>Bit 15 = Active Color RAM Bank for GC 7<br>Bit 14 = Active Color RAM Bank for GC 6<br>Bit 13 = Active Color RAM Bank for GC 5<br>Bit 12 = Active Color RAM Bank for GC 4<br>Bit 11 = Active Color RAM Bank for GC 3<br>Bit 10 = Active Color RAM Bank for GC 2<br>Bit 09 = Active Color RAM Bank for GC 1<br>Bit 08 = Active Color RAM Bank for GC 0 |
| GC_Ins_ActBank | 7:0 | Sets the active Instruction Bank.<br>Bit 07 = Active Instruction RAM Bank for GC 7<br>Bit 06 = Active Instruction RAM Bank for GC 6<br>Bit 05 = Active Instruction RAM Bank for GC 5<br>Bit 04 = Active Instruction RAM Bank for GC 4<br>Bit 03 = Active Instruction RAM Bank for GC 3<br>Bit 02 = Active Instruction RAM Bank for GC 2<br>Bit 01 = Active Instruction RAM Bank for GC 1<br>Bit 00 = Active Instruction RAM Bank for GC 0 |

*Table 2-15:* **OSD GC Data Register (Address Offset 0x00A8)**

| 0x00A8 | OSD GC Data | | R/W |
|---|---|---|---|
| Name | Bits | Description | |
| GC_Data | 31:0 | Graphics Controller Data | |

*Table 2-16:* **OSD Software Reset Register (Address Offset 0x100)**

| 0x0100 | OSD Software_Reset | | R/W |
|---|---|---|---|
| Name | Bits | Description | |
| Soft_Reset_Value | 31:0 | Soft Reset to reset the registers and IP Core, data Value provided by the EDK create peripheral utility. | |

*Table 2-17:* **OSD ISR (Interrupt Status/Clear) Register (Address Offset 0x0220)**

| 0x0220 | ISR – Interrupt Status/Clear | R/W |
|---|---|---|
| Name | Bits | Description |
| OSD GC Address Overflow | 31:24 | OSD GC Instruction Overflow Interrupt<br>Indicates that the HOST tried to write beyond the maximum address for the instruction ram, font ram, text ram or color ram (for the currently selected write bank address).<br>31: Graphics Controller 7<br>…<br>24: Graphics Controller 0 |
| OSD GC_Ins_error | 23:16 | OSD GC Instruction Error Interrupt<br>Indicates that the GC could not complete all instructions. This interrupt is asserted if an END opcode (binary 0000) is not found before the end of each graphics line.<br>23: Graphics Controller 7<br>…<br>16: Graphics Controller 0 |
| Reserved | 15 | RESERVED |
| OSD Input TLAST Late | 14 | OSD Input TLAST Late Interrupt<br>Asserted if any AXI4-Stream input layer TLAST signal happens after the expected clock cycle. |
| OSD Input TLAST Early | 13 | SD Input TLAST Early Interrupt<br>Asserted if any AXI4-Stream input layer TLAST signal happens before the expected clock cycle. |
| Reserved | 15 | RESERVED |
| OSD Input Error | 11:4 | OSD Input Error Interrupt<br>This interrupt is asserted if the input is set to AXI4-Stream and tlast signal is not asserted at the expect cycle. The OSD will detect if the tlast signal is early or late.<br>11: input 7<br>..<br>4: input 0 |
| OSD VBI End | 3 | OSD Vertical Blank Interval End Interrupt |
| OSD VBI Start | 2 | OSD Vertical Blank Interval Start Interrupt |
| OSD Frame Error | 1 | OSD did not complete processing frame before next Vblank |
| OSD Frame Done | 0 | OSD completed processing Frame |

*Table 2-18:* **OSD IER (Interrupt Enable) Register (Address Offset 0x0228)**

| 0x0228 | IER – Interrupt Enable | R/W |
|---|---|---|
| Name | Bits | Description |
| OSD GC Address Overflow | 31:24 | OSD GC Instruction Overflow Interrupt Enable |
| OSD GC_Ins_error | 23:16 | OSD GC Instruction Error Interrupt Enable |
| Reserved | 15 | |
| OSD Input TLAST Late | 14 | OSD Input TLAST Late Interrupt Enable |
| OSD Input TLAST Early | 13 | SD Input TLAST Early Interrupt Enable |
| Reserved | 12 | |
| OSD Input Error | 11:4 | OSD Input Error Interrupt Enable |
| OSD VBI End | 3 | OSD Vertical Blank Interval End Interrupt Enable |
| OSD VBI Start | 2 | OSD Vertical Blank Interval Start Interrupt Enable |
| OSD Frame Error | 1 | OSD Frame Error Interrupt Enable |
| OSD Frame Done | 0 | OSD Frame Done Interrupt Enable |

Chapter 3

# Customizing and Generating the Core

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

## GUI

This section contains details about the CORE Generator™ tool GUI and the EDK GUI.

### CORE Generator Tool GUI

The CORE Generator tool GUI is shown in Figure 3-1 and Figure 3-2. Field descriptions are provided in Global Parameters, page 65. Each field sets a parameter used at build time to configure different hardware options.

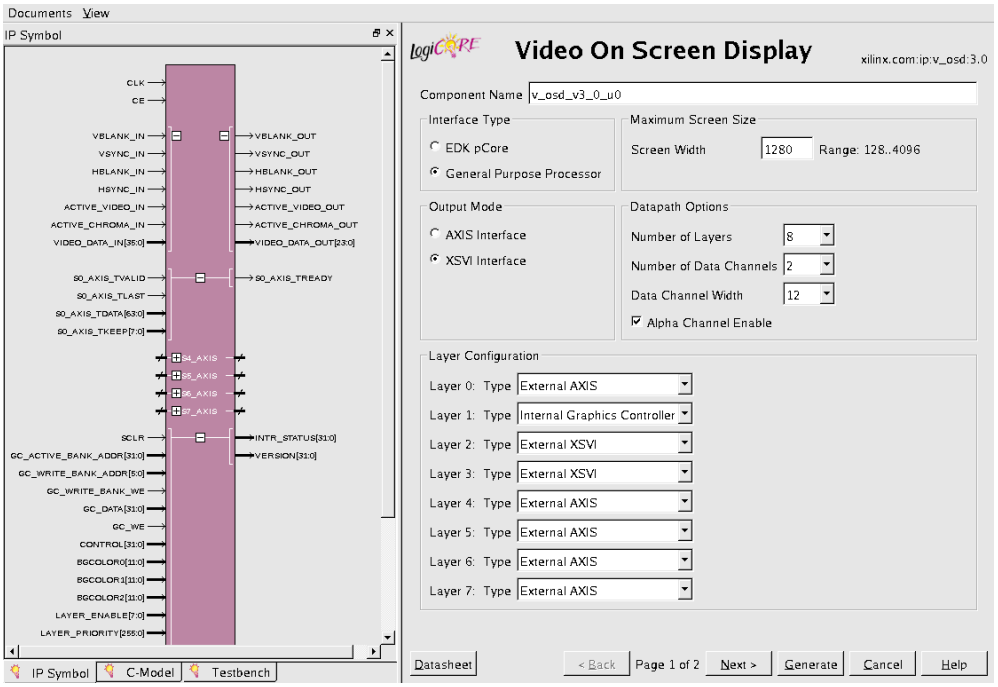



*Figure 3-1:* **CORE Generator GUI - Main Window**

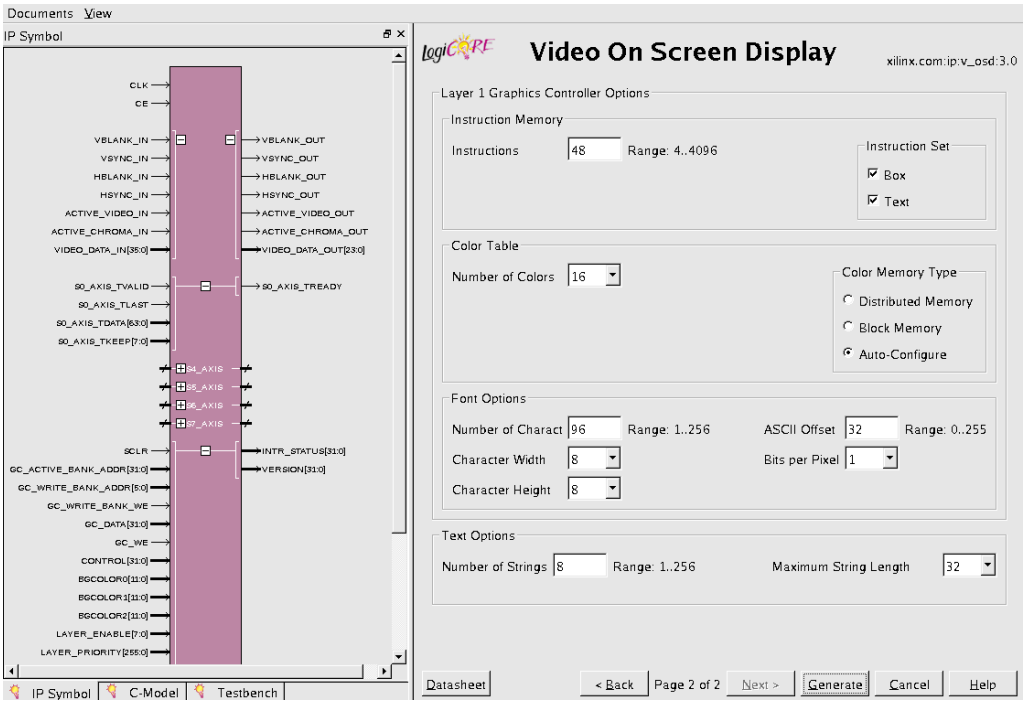



*Figure 3-2:* **CORE Generator GUI - Graphics Controller Options Window**

***Note:*** The Graphics Controller Options Window is available only if the Layer Type is set to “Internal Graphics Controller.”

## Global Parameters

- **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 9and “_”.
- **Interface Type**: The On-Screen Display is generated with one of two control interfaces.
  - EDK pCore Interface: CORE Generator generates the core as a pCore which can be easily imported into an EDK project as a hardware peripheral. The core registers can be programmed in real-time via a MicroBlaze™ processor and the AXI4-Lite Interface. See EDK pCore AXI4-Lite Interface in Chapter 2.
  - General Purpose Processor Interface: the CORE Generator tool generates a set of ports that can be used to program the core. See General Purpose Processor Interface in Chapter 2 .
- **Maximum Screen Size**: This field configures the maximum allowed screen size. The Maximum screen width is configurable. Changing this field affects several counters, comparators and memory (Block RAM) usage. Increased screen size increases resource usage. Valid range for Screen Width is {128 .. 4096}.
- **Output Mode**: This field configures the On-Screen Display for one of two output interfaces.
  - AXI4-Stream Interface: The core is generated with the output AXI4-Stream Master ports enabled. See Output AXI4-Stream Master Interface, page 49.

  - XSVI Interface: The core is generated with the Xilinx Streaming Video Interface port enabled. See Xilinx Streaming Video Interface (XSVI) in Chapter 2.
  - Corresponds to the `C_OUTPUT_MODE` Parameter of the EDK pCore.
- **Number of Layers**: This field configures the number of layers to alpha blend together. Each layer can be configured to read data from the FIFO inputs or from one of the internal Graphics Controllers. Valid range is (1 .. 8). Corresponds to the `C_NUM_LAYERS` Parameter of the EDK pCore.
- **Number of Data Channels**: This field configures the number of data channels. Valid values are 2 and 3.
  - 2 = Two data channels. Typically used for YUV 4:2:2 data. Either channel can be luminance (Y) or chrominance (Cb,Cr) channels if no internal Graphics Controller is used. If using a Graphics Controller layer, the Graphics Controller outputs luminance (Y) on channel 0 and chrominance (Cb,Cr) on channel 1.
  - 3 = Three data channels. Used for RGB and YUV 4:4:4 data. This mode is color component agnostic. Each channel can be configured for any color component.

  Corresponds to the `C_NUM_DATA_CHANNELS` Parameter of the EDK pCore.
- **Data Channel Width**: This field configures the data width of each color component channel. Valid values are 8, 10 and 12. Configuring the Data Channel Width and the Number of Data Channels yields an effective bits per pixel of 16, 20, 24, 30 or 36 bits.
- **Alpha Channel Enable**: This field enables the alpha channel on the XSVI and VFBC Read input data buses. Corresponds to the `C_ALPHA_CHANNEL_EN` Parameter of the EDK pCore.
- **Layer Configuration – Layer # Type**: These fields configure the type, or data source, of each layer, one field for each layer. Each layer is numbered from 0 to 7. The maximum number of layers is set by the Number of Layers field. Three data sources are valid:
  - External AXI4-Stream: This is an input AXI4-Stream slave interface with tdata, tkeep, tvalid, tready and tlast. See Input AXI4-Stream Slave Interface(s), page 49.
  - Internal Graphics Controller: If the layer is configured for this type, then the Read FIFO interface inputs are ignored and all data is generated and read from an internal Graphics Controller.
  - External XSVI: Selecting this data source, will internally connect this layer data bus to the `video_data_in` bus from the XSVI input interface. The XSVI `video_data_in` bus input can be routed to one or more layers.

### Graphics Controller Parameters

- **Instructions**: This field configures the maximum number of Graphics Controller instructions that can be executed per frame. Increasing this number increases the number of Block RAMs utilized.
- **Instruction Set**: This field configures which instructions are valid for the Graphics Controller implementation. Two instructions are currently configurable: box and text. Other instructions, including NoOp, are always available.
- **Number of Colors**: This field configures the size of the color palette used by the Graphics Controller. Valid values are 16 and 256.
- **Color Memory Type**: This field configures how the color palette is implemented in hardware, as Distributed RAM, as Block RAM or Auto-Configured. In auto-configuration mode, distributed RAM will be used if the color palette is small

enough. The RAM type can be overridden if it is known which type is preferred for the application.

- **Number of Characters**: This field configures the number of characters to be stored within the internal Font RAM. Valid values are 1 to 256. This field, along with the Character Width, Character Height, ASCII Offset and Bit per Pixel fields, affects the overall size of the Font RAM.
- **Character Width**: This field configures the width of each character. The width is in pixels. Valid values are 8 and 16.
- **Character Height**: This field configures the height of each character. The height is in video lines. Valid values are 8 and 16.
- **ASCII Offset**: This field configures the ASCII value of the first location in the Font RAM. This is useful if it is known that certain ASCII values will not be used.
- **Bits per Pixel**: This field configures the bits per pixel of each character. Valid values are 1 and 2.
  - 1 = One bit per pixel. This yields a foreground and a background color for each character.
  - 2 = Two bits per pixel. This allows each character pixel to be programmed to one of four different colors.
- **Number of Strings**: This field configures the maximum number of strings to be stored within the Text RAM. This field, along with the Maximum String Length field, affects the overall size of the Text RAM. The maximum number of strings cannot exceed 256.
- **Maximum String Length**: This field configures the maximum string length allowed for each string within the Text RAM. Valid values are 32, 64, 128 and 256.

## EDK pCore GUI

When the OSD core is generated from the CORE Generator software as an EDK pCore, it is generated with each option set to the default value. All customizations of the pCore are done with the EDK pCore GUI. Figure 3-3 illustrates the EDK pCore GUI for the Video On-Screen Display pCore. All of the options in the EDK pCore GUI for the OSD core

correspond to the same options in the CORE Generator software GUI. See CORE Generator Tool GUI, page 64 for details about each option.

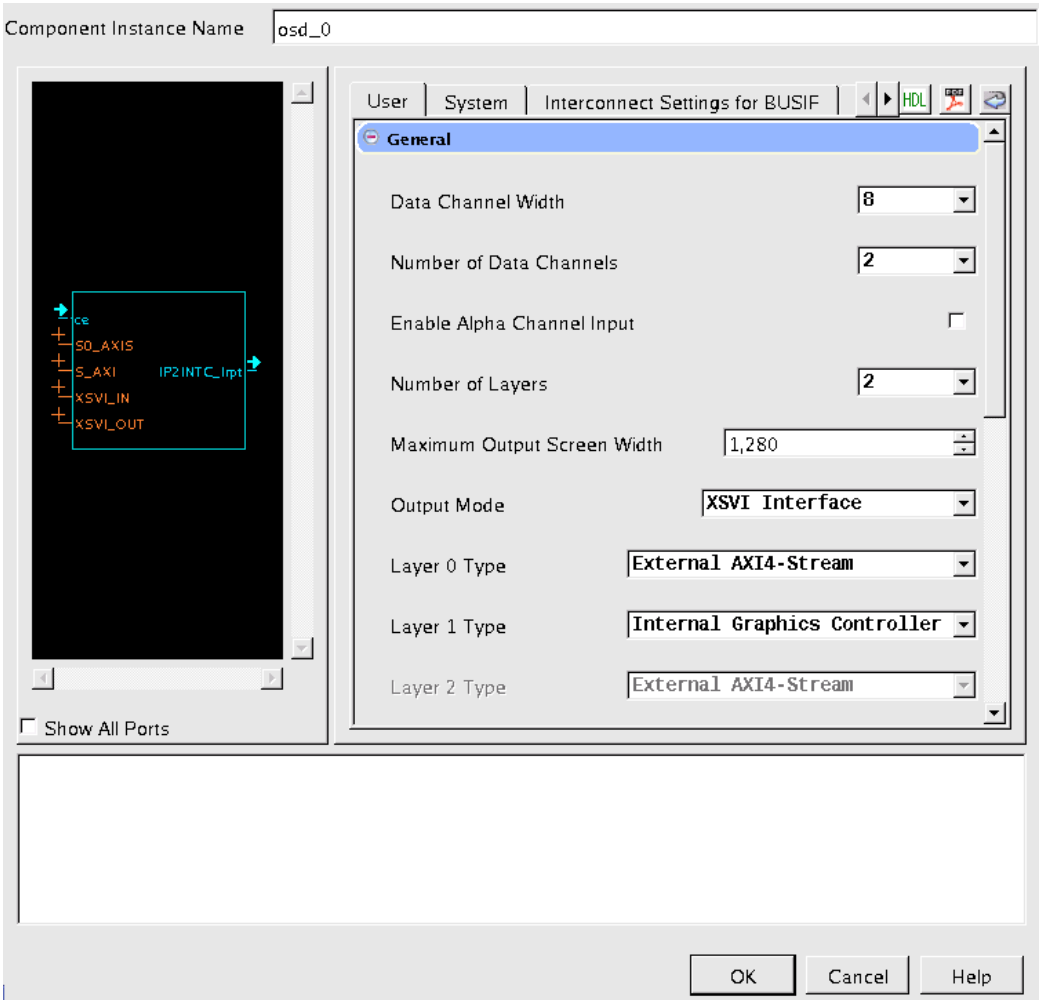



*Figure 3-3:* **EDK GUI**

# Parameter Values in the XCO File

Table 1 defines valid entries for the Xilinx CORE Generator software (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**

| XCO Parameter | Default | Valid Values |
|---|---|---|
| component_name | v_osd_v3_0_u0 | ASCII text using characters: a..z, 0..9 and "_" starting with a letter.<br>***Note:*** "v_osd_v4_0" is not allowed. |
| interface_type | EDK_pCore | EDK_pCore, General_Purpose_Processor |
| data_channel_width | 8 | 8,10,12 |
| number_of_data_channels | 2 | 2,3 |
| alpha_channel_enable | false | true, false |
| number_of_layers | 2 | 1-8 |
| screen_width | 1280 | 128-4096 |
| output_mode | XSVI_Interface | XSVI_Interface, AXIS_Interface |
| layer<#>_box_instruction_enable(1) | true | true, false |
| layer<#>_text_instruction_enable(1) | true | true, false |
| layer<#>_instruction_memory_size(1) | 48 | 4-4096 |
| layer<#>_color_table_memory_type(1) | Auto-Configure | Auto-Configure, Distributed_Memory, Block_Memory |
| layer<#>_color_table_size(1) | 16 | 16,256 |
| layer<#>_font_character_width(1) | 8 | 8,16 |
| layer<#>_font_character_height(1) | 8 | 8,16 |
| layer<#>_font_bits_per_pixel(1) | 1 | 1,2 |
| layer<#>_font_ascii_offset(1) | 32 | 0-255 |
| layer<#>_font_number_of_characters(1) | 96 | 1-256 |
| layer<#>_text_max_string_length(1) | 32 | 32,64,128,256 |
| layer<#>_text_number_of_strings(1) | 8 | 1-256 |
| layer<#>_type=External_AXIS(1) | External_AXIS | External_AXIS, External_XSVI, Internal_Graphics_Controller |

1. <#> is the layer number. The valid values are 0 to 7.

# Parameter Modification in CORE Generator

EDK pCore parameters found in the `<coregen project directory>/v_osd_v3_0_u0/pcores/axi_osd_v3_00_a/data/axi_osd_v3_1_0.mpd` file *cannot* be modified in the Xilinx CORE Generator tool. Parameters shown on the CORE Generator GUI are disabled if the EDK pCore (AXI4-Lite) Interface is selected. Xilinx

recommends that all parameter changes be made with the Video On-Screen Display pCore GUI in the EDK environment.

# Output Generation

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

## EDK pCore Files

When the interface type is set to EDK pCore, CORE Generator then outputs 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 within the `<component_name>` directory:

- drivers
  - osd_v1_03_a
    - data
    - doc
      - html
      - api
    - example
    - src
- pcores
  - axi_osd_v3_00_a
    - data
    - hdl
      - vhdl

## File Details

### <project directory>

This is the top-level directory. It contains xco and other assorted files, as shown in Table 3-2.

*Table 3-2:* **<project directory> 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_osd_v3_00_a/data

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

< project directory>/<component_name>/pcores/axi_osd_v3_00_a/hdl/vhdl

This directory contains the HDL files that implement the pCore.

< project directory>/<component_name>/drivers/osd_v1_03_a/data

This directory contains files that Software Development Kit (SDK) uses to define the operation of the pCore's software driver.

< project directory>/<component_name>/drivers/osd_v1_03_a/doc/html/api

This directory contains HTML documentation files for the pCore's software driver.

< project directory>/<component_name>/drivers/osd_v1_03_a/src

This directory contains the source code of the pCore's software driver, as shown in Table 3-3.

*Table 3-3:* **src Files**

| Name | Description |
|---|---|
| xosd.c | Provides the Application Program Interface (API) access to all features of the Video On-Screen Display device driver. |
| xosd.h | Provides the API access to all features of the Video On-Screen Display device driver. |
| xosd_g.c | Contains a template for a configuration table of Video On-Screen Display core. |
| xosd_hw.h | Contains identifiers and register-level driver functions (or macros) that can be used to access the Video On-Screen Display core. |
| xosd_intr.c | Contains interrupt-related functions of the Video On-Screen Display device driver. |
| xosd_sinit.c | Contains static initialization methods for the Video On-Screen Display device driver. |

## General Purpose Processor Files

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

### File Details

The CORE Generator software output consists of some or all the files shown in Table 3-4.

*Table 3-4:* **GPP Output Files**

| Name | Description |
|---|---|
| <component_name>_readme.txt | Readme file for the core. |
| <component_name>.ngc | The netlist for the core. |

*Table 3-4:* **GPP Output Files *(Cont'd)***

| Name | Description |
|---|---|
| <component_name>.veo<br><component_name>.vho | The HDL template for instantiating the core. |
| <component_name>.v<br><component_name>.vhd | The structural simulation model for the core. It is used for functionally simulating the core. |
| <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. |
| <component_name>.asy | IP symbol file |
| <component_name>.gise<br><component_name>.xise | ISE software subproject files for use when including the core in ISE software designs. |

XILINX®

Chapter 4

# Designing with the Core

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

## General Design Guidelines

The Xilinx LogiCORE™ IP On-Screen Display core reads 2D video image data in raster order from up to eight sources. Each data source can be configured to be an AXI4-Stream, Xilinx Streaming Video Interface (XSVI) or internal graphics controller. If an XSVI or AXI4-Stream interface is selected, ports on the OSD are available for connecting to and reading data from other Xilinx Video IP or from the AXI Video Direct Memory Access Controller (AXI VDMA). These ports are also generic enough for easy integration with any FIFO. If an internal graphics controller is selected to be a source, then the OSD automatically handles interfacing to each graphics controller.

Pixel data from each source is combined using alpha-blending. The resultant output is a 2D video image stream that can be presented to one of two output interfaces – to a FIFO interface or to an AXI4-Stream interface. If an AXI4-Stream interface is used, the `m_axis_tready` and the `s<LAYER_NUM>_axis_tvalid` (from each slave AXI4-Stream video layer input source) will halt operation of the OSD. If an XSVI output is used, the data is presented on the output along with video synchronization signals (Vertical Blank, Horizontal Blank, etc.). The data is presented at the time required by the synchronization signals, and thus, if the AXI4-Stream input(s) are stalled for more than 8 cycles, the video will be corrupted. Each AXI4-Stream input has a small internal FIFO with a depth of 8. Care must be taken to make sure each input FIFO does not underflow.

Only one output interface can be selected. The OSD cannot drive both interfaces at the same time. See EDK pCore AXI4-Lite Interface in Chapter 2 for more information on both the output FIFO (VFBC Write Data Interface) and the XSVI output.

If the OSD is configured for XSVI output, the OSD also requires an XSVI input. This interface is used to receive the horizontal/vertical blank and sync signals as well as an active video signal. The horizontal and vertical sync signals are not used internally to the OSD and are only delayed and presented on the XSVI output for use with driving external display hardware. The same is true for the active video signal and is used only to delineate the presence of valid output. Only the horizontal and vertical blank signals are used internally to the OSD for control and synchronization of frame data. The OSD will only use the video_data on the XSVI input if any video layer is configured for XSVI input. The OSD cannot delay the XSVI input data, but it can crop this input data to the background color.

See DS857, *LogiCORE IP Video Timing Controller Data Sheet*, for more information on video timing signals.

If an XSVI output is used, the use of specific video timing signals requires that the input data is always present when requested (the OSD will not look at the internal input FIFO

empty flags). If an output AXI4-Stream interface is used, it is required that the input and output tvalid and tready ports do not delay the operation of OSD too often, as this may cause frames to not be completely processed. Specific interrupt status bits will alert such errors.

An example OSD configuration with three data sources (layers) is shown in Figure 4-1. Data for layer 0 and layer 1 are read from input FIFOs. Data for layer 2 are read from a graphics controller instance.

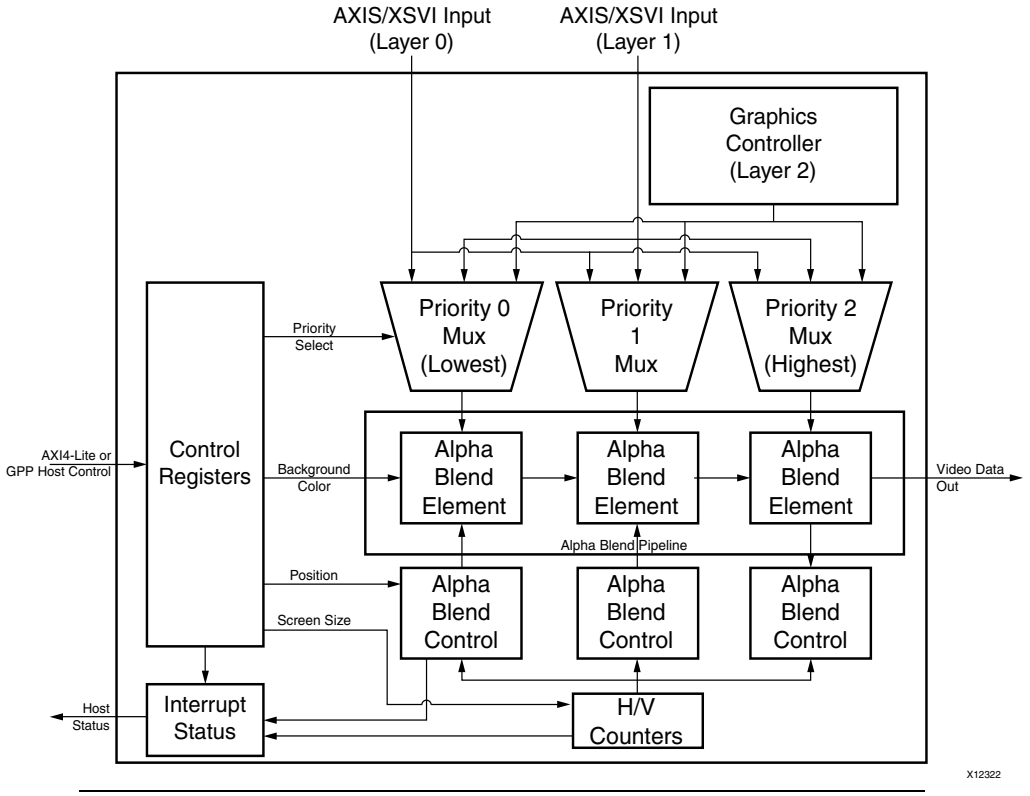



*Figure 4-1:* **Example OSD Block Diagram**

In addition to the video data interfaces, the Xilinx On-Screen Display has a control interface for setting registers that control the background color and screen size. The size, (x,y) position and priority (Z-plane order) of each layer can also be configured. Registers for overriding pixel based alpha values with a global alpha and for enabling/disabling layers are also provided.

All control registers can be set dynamically in real time. The OSD internally double-buffers all control registers every frame. Thus, control registers can be updated without introducing artifacts on screen. In addition, the OSD provides a "Register Update Enable" bit in the control register that allows controlling the timing of the double-buffered register updates for further flexibility.

A 32-bit interrupt status register output is also provided that flags internal errors or general events that may require host processor intervention. Interrupt status bits flag events for vertical blanking start and end, frame error, frame complete, incorrect AXI4-Stream tlast placement, and graphics controller errors (discussed later).

## Alpha-Blending Pipeline

The Xilinx On-Screen Display alpha-blending pipeline includes from one to eight alpha-blending elements connected in succession. Each element blends the pixel data from one layer to the pixel data from the layer underneath, and controls whether a layer is enabled and if pixel-level alpha should be read from the input alpha channel or a global alpha value should be used.

Layer data is blended in the order dictated by the priority setting for each layer in the control registers. The priority values are used to multiplex layer data to the correct alpha-blending element.

A basic flow chart diagram showing the alpha-blending process is shown in Figure 4-2.

The alpha-blending pipeline architecture takes advantage of the high-performance XtremeDSP™ DSP48 slices available in the target device families. These slices are utilized for multiplication and some addition operations and time-shared efficiently between color component channels.

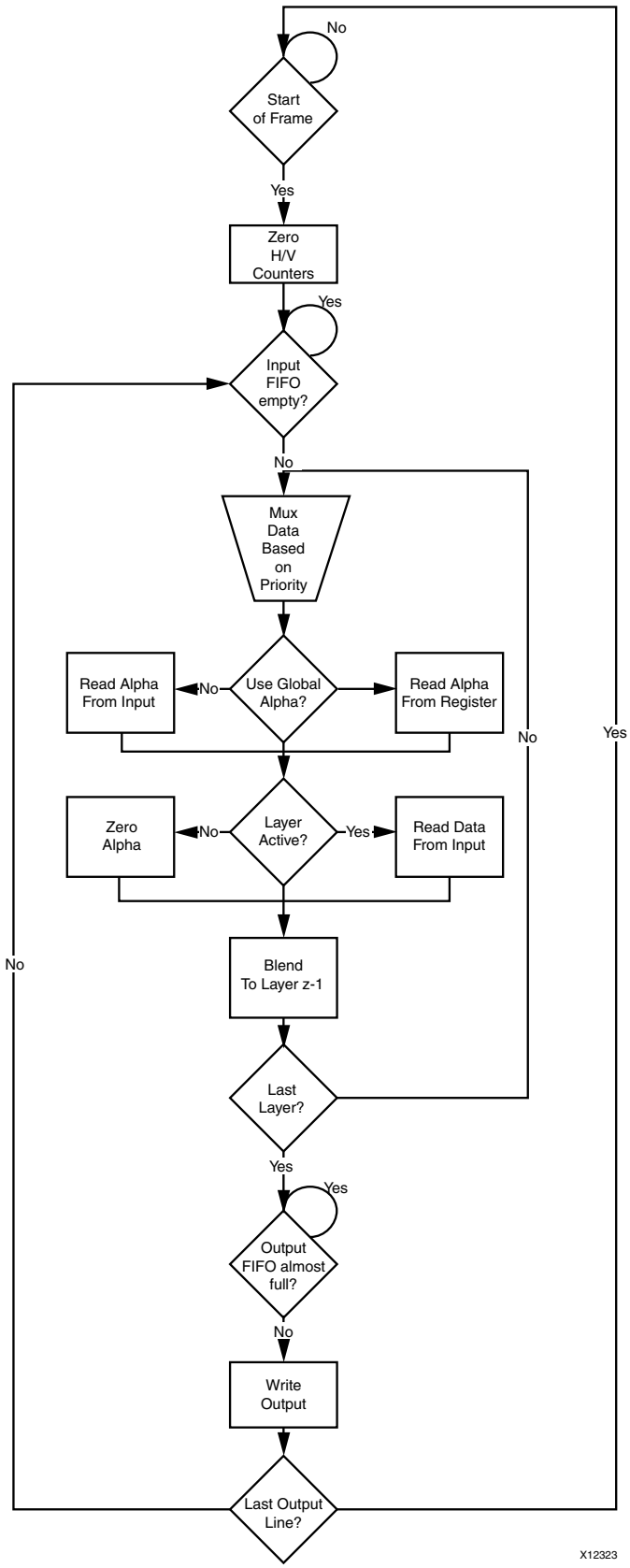



*Figure 4-2:* **Alpha-Blending Pipeline Flow Chart**

## Graphics Controller

The Xilinx On-Screen Display internal graphics controller can generate two graphics elements: boxes and text strings. Boxes can be drawn filled or outlined. The color, position, size and outline weight of each box are configurable via host control registers (graphics controller host interface). Text strings can be drawn with a scale factor of 1x, 2x, 4x, or 8x the original size. The color and position are also configurable.

Figure 4-3 shows the internal structure of the graphics controller.

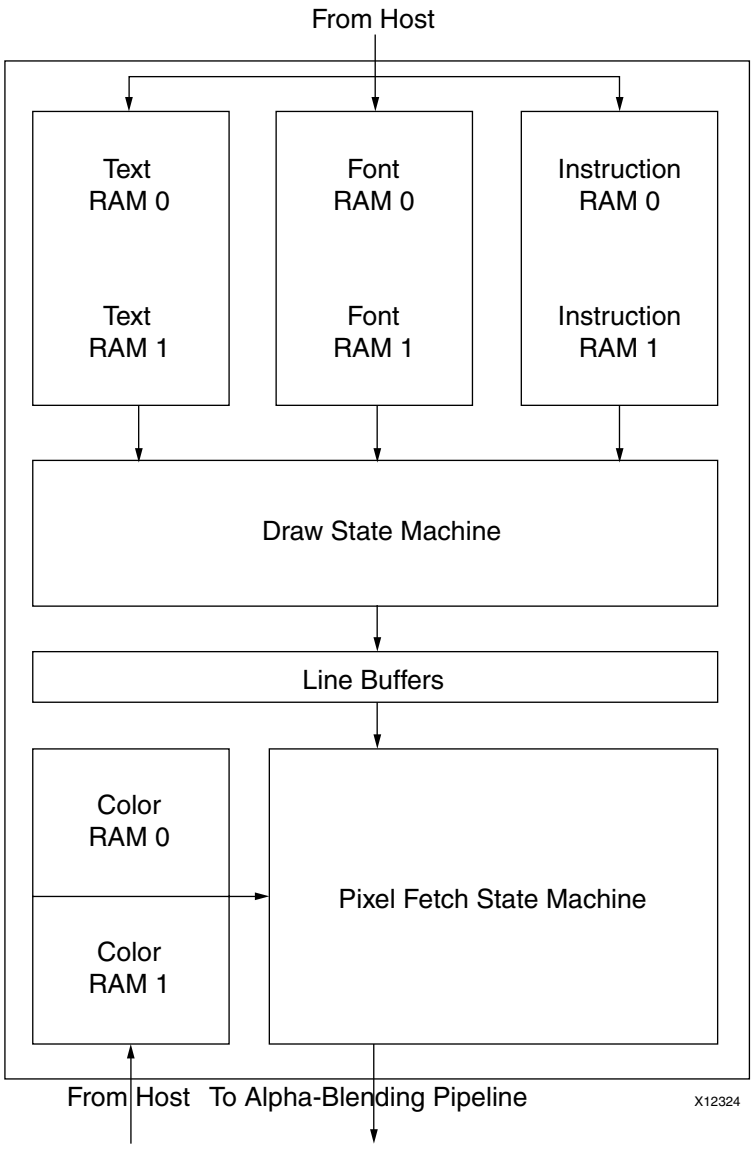



*Figure 4-3:* **OSD Graphics Controller Block Diagram**

The graphics controller is configured to draw boxes and text by a host processor. The host processor must write graphics instructions into an Instruction RAM. Each instruction can configure the graphics controller to draw a box, a text string, a combined box/text graphics element or to perform an internal function. The maximum number of instructions is configured with the "Instructions" field of the CORE Generator™ tool GUI.

During every video line, the draw state-machine fetches instructions from an Instruction RAM and draws multiple graphics elements to a line buffer. A box draw instruction will cause the draw state-machine to draw a box of the selected color to a line buffer. A text draw instruction will cause the draw state-machine to fetch a text string from a Text RAM. This text string is used to fetch character data from a Font RAM. The character data along with the color selected by the instruction is used to write pixels in a line buffer.

The pixel fetch state-machine generates output pixel data. It reads the data in the line buffers and uses this data to select a color from the Color RAM for any given pixel. Output pixel data is generated in real-time in raster order. The color and alpha for each output pixel is decided upon when requested. This eliminates the need for external memory storage. The pixel fetch state-machine never reads from the same line buffer that is being written to by the draw state-machine.

Note that for each memory type (Instruction, Color, Text and Font), there are two memories – RAM 0 and RAM 1. This duplication allows the host processor to write to one memory while the graphics controller is reading from another. This eliminates screen artifacts while the processor is configuring the graphics controller.

Memory boundaries are conceptual only. Some graphics controller memories may be efficiently combined to save Block RAM or Distributed RAM storage.

Each graphics controller has a set of parameters that controls its configuration. These parameters affect the size of each memory and the resources used by the Xilinx On-Screen Display. See Global Parameters in Chapter 3 for more information on the graphics controller parameters.

# Algorithm

This section explains the alpha-blending concept used in the Xilinx On-Screen Display. For more information on the internal structure of the OSD and the Alpha-Blending Pipeline, see Chapter 4, Implementation.

## Alpha-Compositing and Alpha-Blending

Alpha-compositing is the process of combining two images with the appearance of partial transparency. To perform this composition, a matte (or array) is created that contains the coverage information for each pixel within each image. This matte information is typically stored in a channel and transmitted alongside each pixel color. This is referred to as the alpha channel. The alpha channel range of values is from 0 to 1, where “0” represents that the current pixel does not contribute to the final image and is fully transparent. “1” represents that the current pixel is fully opaque. Any value in between represents a partially transparent pixel.

Different algebraic compositing algorithms define different image blending operations. These operations range from “over,” “in,” “out,” “atop,” to “xor” and other logical operations. For this design, the only concern is the “over” operation. The “over” operation describes the combination of one image that resides over another.

Alpha blending is the convex combination of two pixels, allowing for transparency, and describes one subset of the alpha compositing operations—the over alpha-compositing operation. The two pixels to be blended reside within two different image layers. Each layer has a definite Z-plane order. In other words, each layer resides closer or farther from the observer and has a different depth. Thus, the image pixel and the image pixel directly “over” it are to be blended.

The equation for alpha-blending one layer to the layer directly behind in the Z-plane is below. This operation is conceptually simple linear interpolation between each color component of each layer. Since the operation is the same for each color component, this implies that the same hardware could be reused for each color component given a high enough operating frequency.

$$Component'_{(x,y,z)} = \alpha_{(x,y,z)} Component_{(x,y,z)} + (1 - \alpha_{(x,y,z)}) Component_{(x,y,z-1)}$$

Where:

- $\alpha_{(x,y,z)}$ is the alpha value in the range {0.0 .. 1.0} from the alpha channel associated with the pixel at coordinates (x,y) in Layer z.
- $Component_{(x,y,z)}$ represents one color component channel from the color space triplet (RGB, YUV, etc.) associated with the pixel at coordinates (x,y) in Layer z.
- $Component_{(x,y,z-1)}$ represents the same color component at the same (x,y) coordinates in Layer z-1 (one layer below in Z-plane order).
- $Component'_{(x,y,z)}$ is the resulting output component value after alpha-blending the component values from coordinates (x,y) from Layer z and Layer z-1.

The same equation for the next layer above, Layer z+1:

$$Component'_{(x,y,z+1)} = \alpha_{(x,y,z+1)} Component_{(x,y,z+1)} + (1 - \alpha_{(x,y,z+1)}) Component_{(x,y,z)}$$

These alpha-blending operations can be chained together simply by taking the resultant output, $Component'_{(x,y,z)}$, and substituting it into the Layer z+1 equation for $Component_{(x,y,z)}$. This implies that the result of blending Layer z with the background becomes the new background for Layer z+1, or the layer directly over it. In this way, any number of image layers can be blended by taking the blended result of the layer below it. This also implies that the Z-plane order could affect the final result. This is especially true if all alpha values are 1.

Typically, the order in which layers are blended is determined by their priority setting. Each image layer is assigned a priority number. The higher the priority, the more in the foreground it is and the "closer" it is to the observer. Thus, those layers with a higher priority reside on top of layers with a lower priority. This priority is also referred to as the Z-plane order and is real-time configurable.

# Clocking

The Video On-Screen Display core has one clock (`clk`) that is used to clock the entire core, including the AXI interfaces and the core logic.

# Resets

When configured with the GPP Interface, the Video On-Screen Display core has one reset (`sclr`) that is used for the entire core. The reset is active High.

When configured with the AXI4-Lite Interface, the Video On-Screen Display core has one reset (`aresetn`) that is used for the entire core. The reset is active Low.

# Protocol Description

For the pCore version of the Video On-Screen Display core, the register interface is compliant with the AXI4-Lite interface. The input video layer and output interfaces can be configured to be compliant with the AXI4-Stream interface.

XILINX®

Chapter 5

# Constraining the Core

This chapter contains applicable constraints for the Video On-Screen Display core.

## Required Constraints

The `clk` pin should be constrained at the maximum pixel clock rate desired for the video stream.

## Device, Package, and Speed Grade Selections

There are no device, package or speed grade requirements for the Video On-Screen Display core. This core has not been characterized for lower 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.

## Clock Management

There is only one clock, `clk`, for the Video On-Screen Display core. When using the AXI4-Lite EDK pCore interface of the Video On-Screen Display core, the AXI Interconnect core will handle the asynchronous clock domain crossing from the video to the processor clock domain.

## Clock Placement

There are no specific clock placement requirements for this core.

## Banking

There are no specific banking rules for this core.

## Transceiver Placement

There are no transceiver placement requirements for this core.

## I/O Standard and Placement

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

Chapter 6

# Detailed Example Design

This chapter illustrates two usage scenarios for the Xilinx Video On-Screen Display: Multi-FIFO to Stream (XSVI) Mode, page 83 and Multi-FIFO to FIFO Mode, page 84.

## Multi-FIFO to Stream (XSVI) Mode

In this scenario, the Xilinx Video On-Screen Display reads data from multiple input FIFOs and outputs video data in streaming format to an XSVI output. The OSD also reads timing signals from a video timing generator (provided here by the Xilinx Video Timing Controller), which it uses to format the output data.

Input data can be read from any FIFO. In this case, input data is read from external memory using an external memory controller and the Xilinx Video Direct Memory Access. The OSD contains VFBC Read FIFO Interfaces to facilitate connecting to generic FIFO interfaces. The VFBC commands and Command FIFO Interfaces are provided by the VDMA. See Figure 6-1.

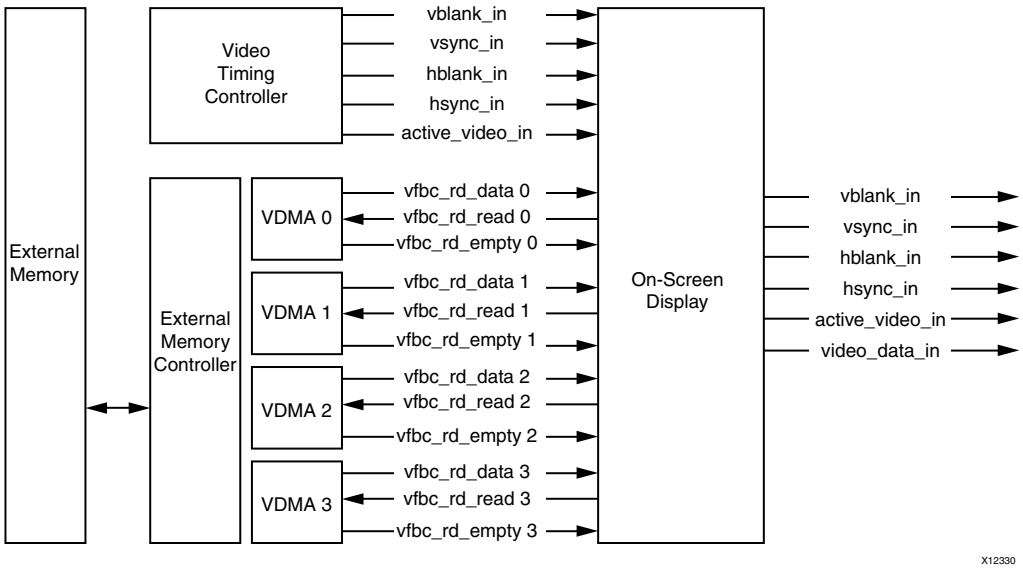



*Figure 6-1:* **Multi-FIFO to Stream Mode**

No FIFO empty flags are used in this example. The specific timing of the streaming output interface requires that the OSD read data from the input FIFOs at specific times, regardless of the FIFO status. The video system should supply input data at a rate sufficient to guarantee the avoidance of FIFO underflow.

For more information on these cores, refer to the following:

- PG020, LogiCORE IP AXI *Video Direct Memory Access (VDMA) Product Guide*
- DS729, *Video Timing Controller Data Sheet*

## Multi-FIFO to FIFO Mode

In this scenario, the Xilinx Video On-Screen Display again reads data from multiple input FIFOs, but instead of outputting data in streaming format to an XSVI output, data is written to an output FIFO interface. The OSD reads timing signals from a video timing generator only to signal the start and stop of a frame, not to format the output data.

Again, input data can be read from any FIFO. In this case input data is read from external memory using an external memory controller and the Xilinx Video Direct Memory Access. The OSD contains VFBC Read and Write FIFO Interfaces to facilitate connecting to generic FIFO interfaces. The VFBC commands and Command FIFO Interfaces are provided by the VDMA. See Figure 6-2.

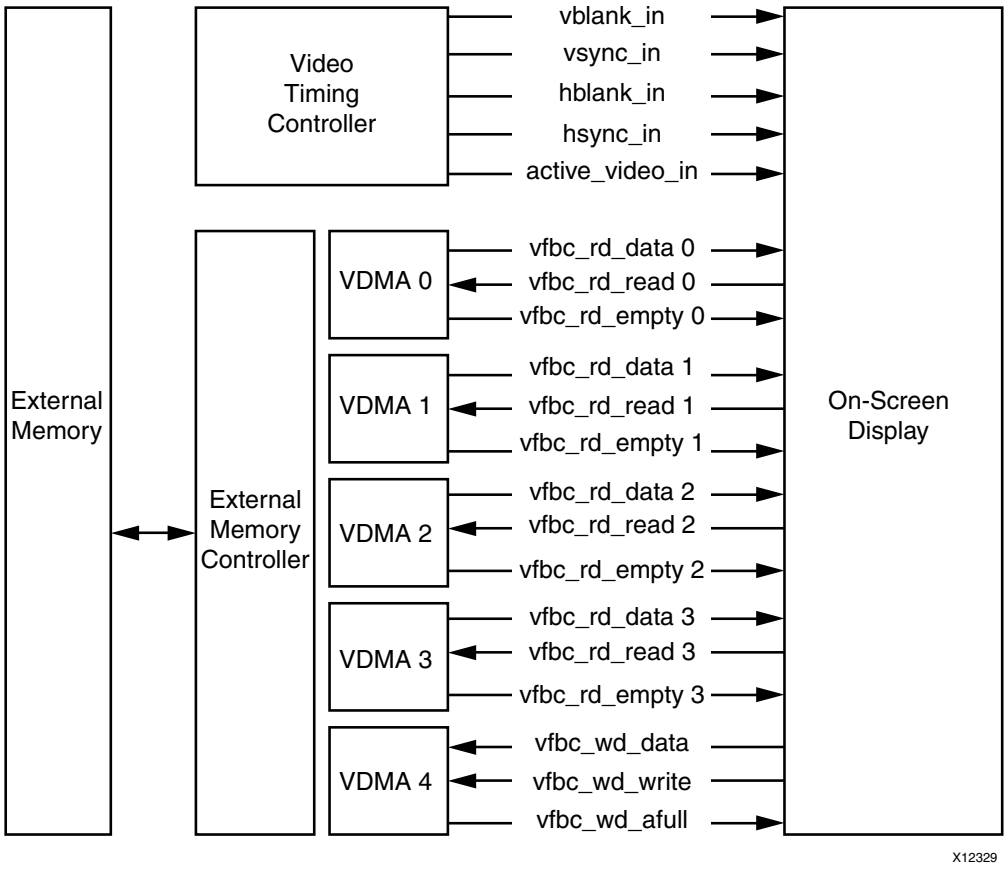



*Figure 6-2:* **Multi-FIFO to FIFO Mode**

FIFO empty flags on the inputs are used in this case. The FIFO almost full flag is also used. The FIFO flags are monitored and will halt the operation of the OSD allowing for simple flow control. The video system should supply enough bandwidth to the OSD FIFO interfaces to guarantee that an entire video frame can be processed before the next video frame is required by the video timing generator.

# Directory and File Contents

The example design contains the following directories and files:

- Expected

  The Expected directory contains the pre-generated expected/golden data used by the test bench to compare to the actual output data.

  - reg_out.txt
  - video_out.txt
- Stimuli

  The Stimuli directory contains the pre-generated input data used by the test bench to simulate the core.

  - `osd.cfg`
  - `reg_in.txt`
  - `video_in0.txt`
  - `video_in1.txt`
  - `video_in2.txt`
  - `video_in3.txt`
  - `video_in4.txt`
  - `video_in5.txt`
  - `video_in6.txt`
  - `video_in7.txt`
- Results

  The Results directory is where the actual simulation output data file is written.
- src

  The src directory contains the .vhd and .xco files of the core.

  - `v_osd_v3_0_u0.vhd`

  The .vhd file is a netlist generated using CORE Generator™ software.

  - `v_osd_v3_0_u0.xco`

  The .xco file can be used with the CORE Generator software to regenerate the netlist.
- tb_src

  The tb_src directory contains the top-level test bench design. This directory also contains other packages used by the test bench.

  - `tb_v_osd_v3_0VHT.vhd`
  - `VHTSimPack.vhd`
  - `v_timebase_v3_0_u0.vhd`
  - `v_timebase_v3_0_u0.xco`
- `isim_wave.wcfg`: Waveform configuration file for iSim
- `mti_wave.do`: Waveform configuration for ModelSim
- `run_isim.bat`: Runscript for iSim in Windows OS
- `run_isim.sh`: Runscript for iSim in Linux OS
- `run_mti.bat`: Runscript for ModelSim in Windows OS
- `run_mti.sh`: Runscript for ModelSim in Linux OS

## Demonstration Test Bench

The demonstration test bench is provided as an introductory package that enables core users to observe the core generated by the CORE Generator tool operating in a waveform simulator. The user is encouraged to observe core-specific aspects in the waveform, make simple modifications to the test conditions, and observe the changes in the waveform.

## Simulation

To start a simulation using ModelSim for Linux, type **source run_mti.sh** from the console.

To start a simulation using ModelSim for Windows, double-click on `run_mti.bat`.

To start a simulation using iSim for Linux, type **source run_isim.sh** from the console.

To start a simulation using iSim for Windows, double-click on `run_isim.bat`.

## Messages and Warnings

Memory collision errors have been observed when running the demonstration test bench. The issue has been investigated, and it has been determined that these errors can be safely ignored. This error message can be suppressed in ModelSim when the global SIM_COLLISION_CHECK option is set to NONE.

XILINX®

*Appendix A*

# *Verification, Compliance, and Interoperability*

This appendix includes information about how the IP was tested for compliance with the protocol to which it was designed.

## Simulation

A highly parameterizable test bench was used to test the Video On-Screen Display core. Testing included the following:

- Register accesses
- Processing of multiple frames of data
- Testing of various frame sizes including 1080p, 720p and 480p
- Varying instantiations of the core
- Varying the data width including 8, 10 and 12
- Varying the number of data channels including 2 and 3
- Varying the number and type of layers including AXI4-Stream and XSVI input interfaces
- Varying the output interface including AXI4-Stream and XSVI output interfaces
- Varying size, location, transparency and over/under of video layers
- Varying the background color
- Varying the number, size, color and transparency of boxes, text generated from the internal graphics controller

## Hardware Testing

The Video On-Screen Display core has been tested in a variety of hardware platforms 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 Interconnect and various other peripherals. The software for the test system included live video input and output for the Video On-Screen Display core. Various tests could be supported by varying the configuration of the Video On-Screen Display core or by loading a different software executable. The MicroBlaze processor was responsible for:
  - Initializing the appropriate input and output buffers in external memory
  - Initializing the Video On-Screen Display core

- Initializing the HDMI/DVI input and output cores for live video
- Launching the test
- Configuring the Video On-Screen Display for various input frame sizes, positions and transparency
- Launching various graphics controller tests for box and text placement, color, size and transparency
- Launching OSD demos including video/graphics resize/movement and on-screen menu demos
- Controlling the peripherals including the UART and AXI VDMAs

XILINX®

Appendix B

# Migrating

This appendix describes migrating from older versions of the IP to the current IP release.

## Migrating to the EDK pCore AXI4-Lite Interface

The Video On-Screen Display v2.0 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:

http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

## Migrating to the AXI4-Stream Interface

The Video On-Screen Display v3.0 changed from the Video Frame Buffer Controller (VFBC) native interfaces to the AXI4-Stream interfaces. As a result, all of the VFBC-related connections have been replaced with an AXI4-Lite interface. For more information, see:

http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

## Parameter Changes in the XCO File

There were no parameter changes in the XCO file.

## Port Changes

The Video On-Screen Display v2.0 and v3.0 changed the port widths of all VFBC interfaces, and added the video_data_in input port on the XSVI interface.

## Functionality Changes

The Video On-Screen Display v2.0 and v3.0 added the ability to drive the video output from one XSVI input video source. This allows overlaying graphics (from Internal Graphics Controller) from live streaming video without the use of external memory.

XILINX®

# Appendix C

# Debugging

Some debugging tips are as follows:

- Verify that the clock pin, `clk`, is connected to the video clock source and is running.
- Verify that reset pin for the AXI4-Lite interface, `s_axi_aresetn`, is active Low and has asserted and deasserted properly.
- Verify that reset pin for the General Purpose Processor interface, `sclr`, is active High and has asserted and deasserted properly.
- Can the Version register be read properly? See Table 2-4, page 54 for register definitions.
- Check the interrupt status register for frame processed/done or specific errors. Check Table 2-17, page 62 and Table 2-18, page 63 for definitions of each bit.
- Verify that the `vblank_in` input is being properly driven.
- For the XSVI output, verify that the `vblank_in`, `hblank_in` and `active_video_in` inputs are properly driven. These are the minimum required port connections for the XSVI output.
  - If a core is connected to the XSVI output that requires one or more of the following output signals, verify that the equivalent XSVI input ports are connected and driven: `vblank_out`, `vsync_out`, `hblank_out`, `hsync_out`, `active_video_out`, and `active_chroma_out`.
- For the AXI4-Stream output interface, verify that the `vblank_in` input is being properly driven. This is the only port from the XSVI input required for the AXI4-Stream output interface.
- Verify that bits 0 and 2 of the Control register are both set to "1". Bit 0 is the Core Enable bit. Bit 2 is the Register Update Enable bit.
- Verify that bits 4 and 5 of the Control Register correspond to the blank polarity of the XSVI input port. The Video On-Screen Display core does not determine the horizontal and vertical blank polarity. Bit 4 is the Input Horizontal Blank Polarity and bit 5 is the Input Vertical Blank Polarity.
- Verify that each input layer has a unique priority number. No two input layers can have the same priority. If two input layers have the same polarity the output will have unexpected results typically manifesting as a horizontal shift in the video by four or more pixels.
- Verify that the Instruction, Color, Font and Text RAMs in the graphics controller(s) have been initialized if any layer is configured to use a graphics controller.
- Verify that each layer size and position does not exceed the output resolution. The Video On-Screen Display does perform range checks, but does not perform error concealment.

- Verify that each box/text size and position does not exceed the output resolution. The Video On-Screen Display does perform range checks and minimal graphics error concealment.
- Verify that each input layer size registers match the actual input size.
- Verify that each input/output data channel match the expected color component.

XILINX®

# Appendix D

# Application Software Development

This chapter includes information about programming the Graphics Controller(s), as well as, controlling the Xilinx Video On-Screen Display via a software driver.

## Programming the Graphics Controller(s)

This section outlines the data format of each Internal Graphics Controller memory and how to program each. To program any of the internal graphics controllers, the host processor must write data into the Instruction RAM, the Color RAM and, if text is enabled, the Font and Text RAMs.

Data can be written before the OSD graphics controller output is enabled or during operation. Each graphics controller contains two of each memory type to allow the host processor to write to one while the other is being used for display. The *Graphics Controller Active Bank Address* register selects which memories are used for display. The *Graphics Controller Write Bank Address* register selects which memory is to be written by the host.

The OSD Graphics Controller RAM is not preloaded with data. The OSD software driver includes example instruction, color, font and string data that can be programmed into the OSD. Multiple Xilinx or user generated instruction, color, font and string data sets can be stored in external memory and written into the OSD to be used during the next video frame after the data is enabled.

### Instruction RAM

The Instruction RAM stores instructions that tell the OSD Graphics Controller to draw objects at programmable locations on the screen. Each instruction is a set of four 32-bit words. The *Instructions* parameter of the CORE Generator™ tool GUI will configure the maximum number of instruction supported by each graphics controller. Table D-1 shows the OSD Graphics Controller instruction format.

*Table D-1:* **OSD Instruction Format**

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | OpCode | | | | Reserved | | | | Xstop (X1) | | | | | | | | | | | | Xstart (X0) | | | | | | | | | | | |
| 1 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | Text Index | | | |
| 2 | Object Size | | | | | | | | Ystop (Y1) | | | | | | | | | | | | Ystart (Y0) | | | | | | | | | | | |
| 3 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | Color Index | | | |

Word 0 contains the instruction opcode and the horizontal start and stop positions. Word 1 is the text index. Word 2 contains the vertical start and stop positions and the objects size.

Word 3 is the color index. Table D-2 through Table D-5 show the specific bit fields for each instruction word.

*Table D-2:* **OSD Instruction Word 0**

| GC Instruction Word 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| OpCode | | | | R | | | | Xstop (X1) | | | | | | | | | | | | Xstart (X0) | | | | | | | | | | | |

| Name | Bits | Description |
|---|---|---|
| OpCode | 31:28 | OSD GC OpCode<br>0000: END<br>0001 – 01111: Reserved<br>1000: NOP<br>1001: Reserved<br>1010: Draw Box<br>1011-1101: Reserved<br>1110: Draw Text<br>1111: Draw Box-Text |
| Reserved | 27:24 | |
| Xstop (X1) | 23:12 | Horizontal end pixel of the object.<br>Starts at pixel 0.<br>Not used with Text (opcodes 1110 or 1111). |
| Xstart (X0) | 11:0 | Horizontal start pixel of the object.<br>Starts at pixel 0. |

*Table D-3:* **OSD Instruction Word 1**

| GC Instruction Word 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| Reserved | | | | | | | | | | | | | | | | | | | | | | | | Text Index | | | | | | | |

| Name | Bits | Description |
|---|---|---|
| Reserved | 31.8 | |
| Text Index | 7:0 | String Index<br>This is the offset into the Text RAM to read the string for the given instruction. |

*Table D-4:* **OSD Instruction Word 2**

| GC Instruction Word 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| Object Size | | | | | | | | YStop (Y1) | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | Ystart (Y0) | | | | | | | | | | | |

| Name | Bits | Description |
|---|---|---|
| Object Size | 31:24 | The Object Size is the Line Width for Boxes and Lines and the Text Scale Factor for Text Boxes.<br>For Opcode 1010 (Draw Box):<br>ObjectSize[7:0] (bits 31:24) = Line Width. The width of the outline of a box or the width of a line. If the line width is set to 0 for a box opcode, the box will be filled instead of outline.<br>For Opcode = 1110 (Draw Text):<br>ObjectSize[7:4] (bits 31:28) = Text scale factor<br>0: Reserved<br>1: 1x text size<br>2: 2x text size<br>4: 4x text size<br>8: 8x text size<br>For Opcode = 1111 (Draw Box-Text):<br>ObjectSize[3:0] = Line Size (bits 27:24)<br>ObjectSize[7:4] = Text Size (bits 31:28) |
| YStop (Y1) | 23:12 | Vertical end line of the object.<br>Starts at line 0.<br>Must be set to the same value as Y0 for text instruction (opcode 1110). |
| YStart (Y0) | 11:0 | Vertical start line of the object.<br>Starts at line 0. |

*Table D-5:* **OSD Instruction Word 3**

| GC Instruction Word 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Color Index | |

| Name | Bits | Description |
|---|---|---|
| Reserved | 31:8 | |
| Color Index | 7:0 | Box/Text Color<br>This is the index into the GC color RAM. This is used as the address to the color RAM. The color RAM must be programmed with the 32bit color for any address used.<br>Currently supporting 16 or 256 colors.<br>For text color, color_index is the background color and (color_index+1) is the foreground color for BPP=1. |

Each instruction word is shown for the General Purpose Processor Interface (little-endian).

## Supported Instructions

This section contains details about the supported instructions.

### Draw Box (Opcode 1010)

This instruction draws a filled or outline box. The following can be configured with the instruction.

- **Position/Size:** The box is drawn from location (X0,Y0) to location (X1,Y1). (X0,Y0) is the upper left-hand corner and (X1,Y1) is the lower-right-hand corner of the box.
- **Color:** The color is set by the Color Index field. Bits [7:0] are used in 256-color mode and bits [3:0] are used in 16-color mode.
- **Line Width:** The draw box instruction reads the ObjectSize field in instruction Word 2 to set the Line Width of outlined boxes. Outline boxes can have a line width of 1 to 255. Box outlines are draw outside the box from (X0,Y0) to (X1,Y1). Setting the line width to zero (0x00) will cause the box to be a filled box from (X0,Y0) to (X1,Y1).

The placement for boxes in YUV-4:2:2 systems may be limited to even horizontal pixel boundaries to avoid color boundary artifacts.

The number of cycles to complete a draw box instruction depends on the Horizontal Start Pixel, Horizontal End Pixel and Object Size settings for the instruction. The number of cycles also depends on the Number of Colors parameter. For the Number of Colors set to 16, then the number of clock cycles will be 10 + (Object_Size + X1 - X0)/8 for each instruction. For the Number of Colors set to 256, then the number of clock cycles will be 10 + (Object_Size + X1 - X0)/4 for each instruction.

### Draw Text (Opcode 1110)

This instruction draws a text string. The following can be configured with the instruction.

- **Position:** The position of the text string is defined by X0 and Y0. (X0,Y0) is the upper left-hand corner of the string. This instruction requires that Y0 and Y1 both be set to the same value. X1 is ignored.

- **Size:** Bits [31:28] of instruction word 2 set the text scale factor. Text can be drawn 1x, 2x, 4x or 8x the size stored internally. This allows for large text strings to be drawn with minimal memory storage. The text string is scaled according to "nearest-neighbor" interpolation.
- **Color:** The color is set by the Color Index field. Bits [7:0] are used in 256-color mode and bits [3:0] are used in 16-color mode. See the Font RAM and Text RAM sections to see how the color index is used along with the Font and Text RAM entries to set the color of each pixel of text.

### Draw Box-Text (Opcode 1111)

This instruction draws a combined box and text string. The box is located at (X0,Y0) to (X1,Y1) and the text is drawn starting at location (X0, Y1+4), just below the lower-left-hand corner of the box. The following can be configured with the instruction:

- **Box Position/Size:** The box is drawn from location (X0,Y0) to location (X1,Y1). (X0,Y0) is the upper left-hand corner and (X1,Y1) is the lower right-hand corner of the box.
- **Text Position:** The position of the text string is defined by X0 and Y1. (X0,Y1+4) is the upper-left-hand corner of the string.
- **Text Size:** Bits [31:28] of instruction word 2 set the text scale factor. Text can be drawn 1x, 2x, 4x or 8x the size stored internally. This allows for large text strings to be drawn with minimal memory storage. The text string is scaled according to "nearest-neighbor" interpolation.
- **Box Line Width:** Bits [27:24] of instruction word 2 set the line width as in the draw box instruction, but the outline can be only from 1 to 16 pixels in weight.
- **Color:** The color is set by the Color Index field and is used to set both the box color and the text color. Bits [7:0] are used in 256-color mode and bits [3:0] are used in 16-color mode. See the Font RAM and Text RAM sections to see how the color index is used along with the Font and Text RAM entries to set the color of each pixel of text.

### NOP (Opcode 1000)

This instruction simply tells the graphics controller to do nothing for this instruction. It is available to allow the host processor to easily manipulate instruction lists. For example, the host processor may maintain a copy of the instruction list in an array in external memory. Instructions can be replaced with the NOP instruction to easily remove instructions from the list without shortening the array.

### END (Opcode 0000)

This instruction tells the graphics controller to stop processing.

The graphics controller operates on an instruction list stored within the Instruction RAM. Each instruction list is simply a list of instructions with the last instruction in the list set to the END instruction (opcode 0000). The instruction list is executed each line and must end with an END instruction to properly terminate processing.

Table D-6 shows an example graphics controller instruction list.

*Table D-6:* **Example Graphics Controller Instruction List (2 Boxes and 1 Box-Text)**

| Address | Data[31:0] | Description |
|---|---|---|
| 0x00 | 0xA005_E050 | **Instruction 0:** Draw Box. X0=80, X1=94. |
| 0x01 | 0x0000_0000 | Text Index. Don't care for this instruction. |
| 0x02 | 0x001f_f0ff | Fill box. Y0=255, Y1=511. |
| 0x03 | 0x0000_0005 | Color Index is 5. Box will be drawn with color in address 5 of the Color RAM. |
| 0x04 | 0xA013_7120 | **Instruction 1:** Draw Box. X0=288, X1=311. |
| 0x05 | 0x0000_0000 | Text Index. Don't care for draw box instruction. |
| 0x06 | 0x042f_f2f0 | Outline box. Line size of 4. Y0=752, Y1=767. |
| 0x07 | 0x0000_000f | Color Index is 15. Box will be drawn with color in address 15 of the Color RAM. |
| 0x08 | 0xf002_0010 | **Instruction 2:** Draw Box-Text. X0=16, X1=32. |
| 0x09 | 0x0000_0007 | Text Index is 7. Text will be drawn with ASCII text string from location 7 in Text RAM. |
| 0x0A | 0x2405_0040 | Text Box of size 2x (zoom text by 2).<br>Line size of 4.<br>Y0=64, Y1=80 |
| 0x0B | 0x0000_0004 | Color Index is 4. Box will be drawn with color in address 3 of the Color RAM. Text will be draw with colors 4 and 5 for 1-bit per pixel text and colors 4, 5, 6 and 7 for 2-bits per pixel. |
| 0x0C | 0x8000_0000 | **Instruction 3:** NOP |
| 0x0D | 0xXXXX_XXXX | Don't Care. |
| 0x0E | 0xXXXX_XXXX | Don't Care. |
| 0x0F | 0xXXXX_XXXX | Don't Care. |
| 0x10 | 0x0000_0000 | **Instruction 4:** Instruction End Instruction List. |
| 0x11 | 0x0000_0000 | Zero fill word 1 for END OpCode. |
| 0x12 | 0x0000_0000 | Zero fill word 2 for END OpCode. |
| 0x13 | 0x0000_0000 | Zero fill word 3 for END OpCode. |

To write the previous example data into the Instruction RAM and to program the OSD to use this data, the host processor must first select Instruction RAM 0 or 1 by writing to the *GC Write Bank Address* register (address offset 0x00A0). Once the Instruction RAM is selected, the host processor must write the data found in Table D-6 to the *GC Data* register (address offset 0x00A8). All data is written to the same OSD register address. Once all the instruction data is written, the host processor can enable the Instruction RAM by writing to the *GC Active Bank Address* register (address offset 0x00A4). This will cause the OSD to use the new instruction list during the next video frame.

Table D-7 shows the OSD register addresses and the data written by the host processor. This example assumes that the host is configuring Instruction RAM 1 of the Graphics Controller on layer 2.

*Table D-7:* **Example OSD Instruction List (2 Boxes 1 TextBox)**

| Address Offset | Data[31:0] | Description |
|---|---|---|
| 0x00A0 | 0x0000_0201 | Sets the Graphics Controller Number to 2 and selects Instruction RAM 1. |
| 0x00A8 | 0xA005_E050 | **Instruction 0:** Draw Box.. (x_start,x_stop) = (80, 94). |
| 0x00A8 | 0x0000_0000 | Text Index. Don't care for instruction A (box draw). |
| 0x00A8 | 0x001f_f0ff | Fill box. (y_start, y_stop) = (255, 511) |
| 0x00A8 | 0x0000_0005 | Color Index is 5.<br>Lookup color set in address 5 of the internal color LUT BRAM. |
| 0x00A8 | 0xA013_7120 | Draw Box. GC#=0. (x_start,x_stop) = (288, 311) |
| 0x00A8 | 0x0000_0000 | Text Index. Don't care for instruction A (box draw). |
| 0x00A8 | 0x042f_f2f0 | Outline box. Line size of 4. (y_start, y_stop) = (752, 767) |
| 0x00A8 | 0x0000_000f | Color Index is 15.<br>Lookup color set in address 15 of the internal color LUT BRAM. |
| 0x00A8 | 0xf002_0010 | Draw BoxText starting at pixel (x_start,x_stop) = (16, 32) |
| 0x00A8 | 0x0000_0007 | Text Index is 7.<br>Lookup the ASCII text string from location 7 in BRAM. |
| 0x00A8 | 0x2405_0040 | Text Box of size 2x (zoom text by 2). Line size of 4. (y_start, y_stop) = (64, 80) |
| 0x00A8 | 0x0000_0003 | Color Index is 3.<br>Lookup color set in address 3 of the internal color LUT BRAM. |
| 0x00A8 | 0x8000_0000 | No-OP. |
| 0x00A8 | X | Don't Care. |

*Table D-7:* **Example OSD Instruction List (2 Boxes 1 TextBox) *(Cont'd)***

| Address Offset | Data[31:0] | Description |
|---|---|---|
| 0x00A8 | X | Don't Care. |
| 0x00A8 | X | Don't Care. |
| 0x00A8 | 0x0000_0000 | End Instruction List Instruction. |
| 0x00A8 | 0x0000_0000 | Zero fill word 1 for END OpCode. |
| 0x00A8 | 0x0000_0000 | Zero fill word 2 for END OpCode. |
| 0x00A8 | 0x0000_0000 | Zero fill word 3 for END OpCode. |
| 0x00A4 | 0x0000_0004 | Sets Instruction RAM 1 of Graphics controller 2 active. |

Host processor writes are shown for the General Purpose Processor Interface (little-endian).

Writing to the *GC Active Bank Address* register will affect all selected memories for all Graphics Controllers. A read-modify-write operation is best performed on this register to avoid incorrectly selecting an invalid memory.

Boxes and text strings that overlap have a distinct Z-plane order and are neither mixed nor alpha-blended. Instructions are performed in the order written into the instruction RAM. Thus, those instructions written later will appear drawn on top of previous instructions.

## Color RAM

The Color RAM can be configured to store 16 or 256 colors. Color RAM data is always written by the host processor in 32-bit data words. The color RAM internal storage is 32-bits wide when the "Data Channel Width" parameter is set to 8 and 64-bits wide when the "Data Channel Width" parameter is set to 10 or 12. Table 7-8 and 7-9 is shown with the "Data Channel Width" parameter set to 8. Table 7-10 and 7-11 is shown with the "Data Channel Width" parameter set to 10.

Within each data location, bits [7:0] contain the channel 0 color component, bits [15:8] contain the channel 1 color component and bits [23:16] contain the channel 2 color component. Bits [31:24] contain the alpha channel value for that color. The storage format is the same for 16 or 256 colors. The number of colors is configured by the "Number of Colors" parameter of the CORE Generator tool GUI.

Table D-8 shows an example Color RAM and its contents with the "Number of Colors" parameter set to 16 and the "Data Channel Width" parameter set to 8.

*Table D-8:* **Example Color RAM Memory Map (16-Colors , 8-bit Data Channel Width)**

| Address | Data[31:0] | Description |
|---|---|---|
| 0x00 | 0x00000000 | Color 0 – 100% Transparent |
| 0x01 | 0x800000ff | Color 1 – Light Red, 50% transparent |
| 0x02 | 0x8000ff00 | Color 2 – Light Green, 50% transparent |
| 0x03 | 0x80ff0000 | Color 3 – Light Blue, 50% transparent |
| 0x04 | 0x80ffff00 | Color 4 – Light Cyan, 50% transparent |

*Table D-8:* **Example Color RAM Memory Map (16-Colors , 8-bit Data Channel Width) *(Cont'd)***

| Address | Data[31:0] | Description |
|---|---|---|
| 0x05 | 0x8000ffff | Color 5 – Light Yellow, 50% transparent |
| 0x06 | 0x80ff00ff | Color 6 – Light Purple, 50% transparent |
| 0x07 | 0x80ffffff | Color 7 – White, 50% transparent |
| 0x08 | 0x80000000 | Color 8 – Black, 50% transparent |
| 0x09 | 0x80000080 | Color 9 – Dark Red, 50% transparent |
| 0x0a | 0x80008000 | Color 10 – Dark Green, 50% transparent |
| 0x0b | 0x80800000 | Color 11 – Dark Blue, 50% transparent |
| 0x0c | 0x80000000 | Color 12 – Black, 50% transparent |
| 0x0d | 0x80808080 | Color 13 – Dark Grey – 50% transparent |
| 0x0e | 0x80c0c0c0 | Color 14 – Light Grey – 50% transparent |
| 0x0f | 0x00000000 | Color 15 – 100% Transparent |

The color descriptions are assuming that the data values are for the RGBA color space with Red on Channel 0, Green on Channel 1 and Blue on Channel 2, but the Color RAM could be configured for any color space.

To write the previous example data into the Color RAM and to program the OSD to use this data, the host processor must first select Color RAM 0 or 1 by writing to the GC *Write Bank Address* register (address offset 0x00A0). Once the Color RAM is selected, the host processor must write the data found in Table D-6 to the *GC Data register* (address offset 0x00A8). All data is written to the same OSD register address. Once all the color data is written, the host processor can enable the Color RAM by writing to the *GC Active Bank Address* register (address offset 0x00A4). This will cause the OSD to use the new color data during the next video frame.

Table D-9 shows the OSD register addresses and the data written by the host processor. This example assumes that the host is configuring Color RAM 1 of the Graphics Controller on layer 2.

*Table D-9:* **Example Color RAM Host Processor Writes (8-bit Data Channel Width)**

| Address Offset | Data[31:0] | Description |
|---|---|---|
| 0x00A0 | 0x00000203 | Sets the Graphics Controller Number to 2 and selects Color RAM 1. |
| 0x00A8 | 0x00000000 | Color data for Color RAM address 0x00 (Color 0) |
| 0x00A8 | 0x800000ff | Color data for Color RAM address 0x01 (Color 1) |
| 0x00A8 | 0x8000ff00 | Color data for Color RAM address 0x02 (Color 2) |
| 0x00A8 | 0x80ff0000 | Color data for Color RAM address 0x03 (Color 3) |
| 0x00A8 | 0x80ffff00 | Color data for Color RAM address 0x04 (Color 4) |
| 0x00A8 | 0x8000ffff | Color data for Color RAM address 0x05 (Color 5) |
| 0x00A8 | 0x80ff00ff | Color data for Color RAM address 0x06 (Color 6) |
| 0x00A8 | 0x80ffffff | Color data for Color RAM address 0x07 (Color 7) |
| 0x00A8 | 0x80000000 | Color data for Color RAM address 0x08 (Color 8) |
| 0x00A8 | 0x80000080 | Color data for Color RAM address 0x09 (Color 9) |
| 0x00A8 | 0x80008000 | Color data for Color RAM address 0x0A (Color 10) |
| 0x00A8 | 0x80800000 | Color data for Color RAM address 0x0B (Color 11) |
| 0x00A8 | 0x80000000 | Color data for Color RAM address 0x0C (Color 12) |
| 0x00A8 | 0x80808080 | Color data for Color RAM address 0x0D (Color 13) |
| 0x00A8 | 0x80c0c0c0 | Color data for Color RAM address 0x0E (Color 14) |
| 0x00A8 | 0x00000000 | Color data for Color RAM address 0x0F (Color 15) |
| 0x00A4 | 0x00000400 | Sets Color RAM 1 of Graphics controller 2 active. |

Host processor writes are shown for the General Purpose Processor Interface (little-endian).

Writing to the *GC Active Bank Address* register will affect all selected memories for all Graphics Controllers. A read-modify-write operation is best performed on this register to avoid incorrectly selecting an invalid memory.

Table D-10 shows an example Color RAM and its contents with the "Number of Colors" parameter set to 16 and the "Data Channel Width" parameter set to 10. The storage format is similar to the 8-bit data channel case, but the color data is 64 bits wide instead of 32. Setting each color requires two data writes to the GC Data register.

*Table D-10:* **Example Color RAM Memory Map (16-Colors, 10-bit Data Channel Width)**

| Address | Data[63:0] | Description |
|---|---|---|
| 0x00 | 0x00000000_00000000 | Color 0 – 100% Transparent |
| 0x01 | 0x00000080_000003ff | Color 1 – Light Red, 50% transparent |
| 0x02 | 0x00000080_000ffc00 | Color 2 – Light Green, 50% transparent |
| 0x03 | 0x00000080_3ff00000 | Color 3 – Light Blue, 50% transparent |
| 0x04 | 0x00000080_3ffffc00 | Color 4 – Light Cyan, 50% transparent |
| 0x05 | 0x00000080_000fffff | Color 5 – Light Yellow, 50% transparent |
| 0x06 | 0x00000080_3ff003ff | Color 6 – Light Purple, 50% transparent |
| 0x07 | 0x00000080_ffffffffff | Color 7 – White, 50% transparent |
| 0x08 | 0x00000080_00000000 | Color 8 – Black, 50% transparent |
| 0x09 | 0x00000080_00000200 | Color 9 – Dark Red, 50% transparent |
| 0x0a | 0x00000080_00080000 | Color 10 – Dark Green, 50% transparent |
| 0x0b | 0x00000080_20000000 | Color 11 – Dark Blue, 50% transparent |
| 0x0c | 0x00000080_00000000 | Color 12 – Black, 50% transparent |
| 0x0d | 0x00000080_20080200 | Color 13 – Dark Grey – 50% transparent |
| 0x0e | 0x00000080_300c0300 | Color 14 – Light Grey – 50% transparent |
| 0x0f | 0x00000000_00000000 | Color 15 – 100% Transparent |

Table D-11 shows the OSD register addresses and the data written by the host processor. This example assumes that the host is configuring Color RAM 1 of the Graphics Controller on layer 2 and that the "Data Channel Width" has been set to 10.

*Table D-11:* **Example Color RAM Host Processor Writes (10-bit Data Channel Width)**

| Address Offset | Data[31:0] | Description |
|---|---|---|
| 0x00A0 | 0x00000203 | Sets the Graphics Controller Number to 2 and selects Color RAM 1. |
| 0x00A8 | 0x00000000 | Color lower data for Color RAM address 0x00 (Color 0) |
| 0x00A8 | 0x00000000 | Color upper data for Color RAM address 0x00 (Color 0) |
| 0x00A8 | 0x000003ff | Color lower data for Color RAM address 0x01 (Color 1) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x01 (Color 1) |
| 0x00A8 | 0x000ffc00 | Color lower data for Color RAM address 0x02 (Color 2) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x02 (Color 2) |
| 0x00A8 | 0x3ff00000 | Color lower data for Color RAM address 0x03 (Color 3) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x03 (Color 3) |
| 0x00A8 | 0x3ffffc00 | Color lower data for Color RAM address 0x04 (Color 4) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x04 (Color 4) |
| 0x00A8 | 0x000fffff | Color lower data for Color RAM address 0x05 (Color 5) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x05 (Color 5) |
| 0x00A8 | 0x3ff003ff | Color lower data for Color RAM address 0x06 (Color 6) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x06 (Color 6) |
| 0x00A8 | 0xffffffffff | Color lower data for Color RAM address 0x07 (Color 7) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x07 (Color 7) |
| 0x00A8 | 0x00000000 | Color lower data for Color RAM address 0x08 (Color 8) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x08 (Color 8) |
| 0x00A8 | 0x00000200 | Color lower data for Color RAM address 0x09 (Color 9) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x09 (Color 9) |
| 0x00A8 | 0x00080000 | Color lower data for Color RAM address 0x0a (Color 10) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x0a (Color 10) |
| 0x00A8 | 0x20000000 | Color lower data for Color RAM address 0x0b (Color 11) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x0b (Color 11) |
| 0x00A8 | 0x00000000 | Color lower data for Color RAM address 0x0c (Color 12) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x0c (Color 12) |
| 0x00A8 | 0x20080200 | Color lower data for Color RAM address 0x0d (Color 13) |
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x0d (Color 13) |
| 0x00A8 | 0x300c0300 | Color lower data for Color RAM address 0x0e (Color 14) |

*Table D-11:* **Example Color RAM Host Processor Writes (10-bit Data Channel Width) *(Cont'd)***

| Address Offset | Data[31:0] | Description |
|---|---|---|
| 0x00A8 | 0x00000080 | Color upper data for Color RAM address 0x0e (Color 14) |
| 0x00A8 | 0x00000000 | Color lower data for Color RAM address 0x0f (Color 15) |
| 0x00A8 | 0x00000000 | Color upper data for Color RAM address 0x0f (Color 15) |
| 0x00A4 | 0x00000400 | Sets Color RAM 1 of Graphics controller 2 active. |

## Font RAM

The Font RAM can be configured to store fixed-distance fonts. The font can be configured either as 8-pixels wide and 8-pixels tall or as 16-pixels wide and 16-pixels tall. In addition, the font can be configured for 1-bit per pixel or for 2-bits per pixel color depth.

Figure D-1 shows an example character (the capital letter 'A') and the corresponding data in the Font RAM to represent that character when the graphics controller is configured for an 8x8 1-bit per pixel font. This example has 8-bits per character line.

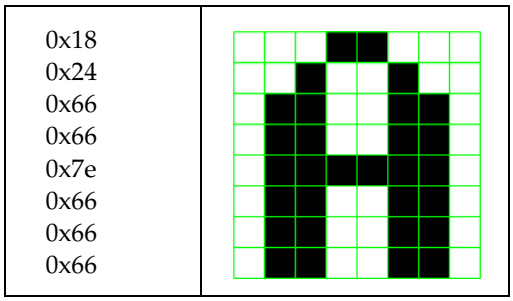



*Figure D-1:* **8x8 1-bit per Pixel Font Example**

When the graphics controller executes a text draw instruction, it uses the pixel data of each character along with the color index of the instruction to set the color of each pixel on screen. In 16-color mode, the graphics controller must set a 4-bit value (bits [3:0]) for each pixel of the character. his is the address used to select the color in the Color RAM (called the color address). The graphics controller will take bits [3:1] of the color address from the color index of the instruction and bit [0] from each bit in the font. Each bit in the font is negated before being used.

For example, if the Color RAM is programmed as shown in Table D-8 for 16-colors, and the current text draw instruction selects color 8 for a string containing the letter 'A', then the 'A' will be drawn as black text (color 8) on a dark-red background (color 9) with 50% transparency.

In 256-color mode, the graphics controller will take bits [7:1] of the color address from the color index of the instruction and bit [0] from each bit in the font.

Figure D-2 shows an example character (a small movie camera icon) and the corresponding data in the Font RAM to represent that character when the graphics

controller is configured for a 16x16 2-bit per pixel font. This example has 32-bits per character line.

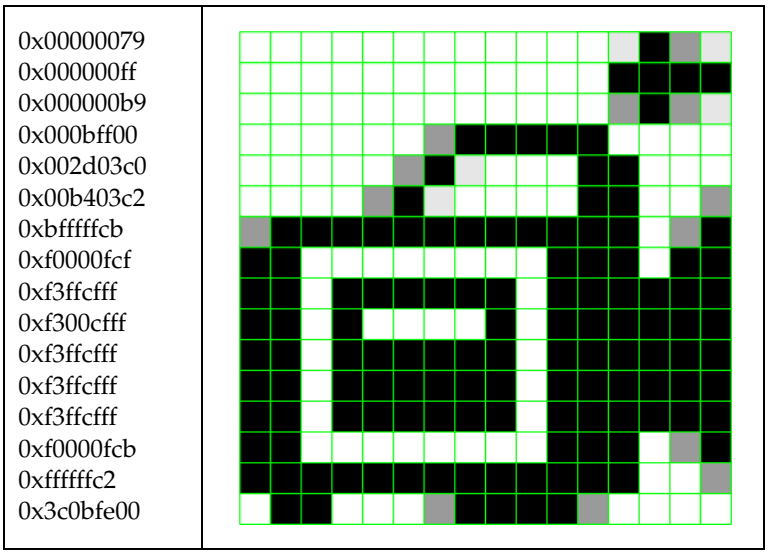



*Figure D-2:* **16x16 2-bits per Pixel Font Example**

In 16-color mode, the graphics controller will set the 4-bit color address (bits [3:0]) for each pixel of the character by taking bits [3:2] from the color index of the instruction and bits [1:0] from the font. Again, each bit in the font is negated before being used.

For example, if the Color RAM is programmed as shown in Table D-8, and the current text draw instruction selects color 12 for a string containing the icon character, then the icon will be drawn as shown in Figure D-2 with black, dark grey, light grey and transparent pixels. Here each set of 2-bits in the font data represents one pixel. “00” is transparent (color 15), “01” is light grey (color 14), “10” is dark grey (color 13) and “11” is black (color 12).

Table D-12 shows an example Font RAM and its contents when the graphics controller is configured for an 8x8 1-bit per pixel font with 96 characters (ASCII 32 to 127). Each 32-bit word represents 4 lines of each character. This example also assumes that the ASCII offset parameter has been set to 32. The ASCII offset is the ASCII value of the first location in memory. Here the first location holds data for the space character (ASCII 32).

*Table D-12:* **Font RAM Memory Map**

| Address | Data[31:0] | Description |
|---|---|---|
| 0x00 | 0x00000000 | Character ' ' (space). Lines 0-3. |
| 0x01 | 0x00000000 | Character ' ' (space). Lines 4-7. |
| 0x02 | 0x18181800 | Character '!'. Lines 0-3. |
| 0x03 | 0x00180018 | Character '!'. Lines 4-7. |
| 0x04 | 0x66666600 | Character '"' (double-quotes). Lines 0-3. |
| 0x05 | 0x00000000 | Character '"' (double-quotes). Lines 4-7. |
| … | | |
| 0x20 | 0x6e663c00 | '@'. Lines 0-3. |
| 0x21 | 0x003e606e | Character '@'. Lines 4-7. |
| 0x22 | 0x663c1800 | Character 'A'. Lines 0-3. |
| 0x23 | 0x00667e66 | Character 'A'. Lines Character 4-7. |
| 0x24 | 0x7c667c00 | Character 'B'. Lines 0-3. |
| 0x25 | 0x007c6666 | Character 'B'. Lines 4-7. |
| … | | |
| 0x5a | 0x04101050 | Character '}'. Lines 0-3. |
| 0x5b | 0x00501010 | Character '}'. Lines 4-7. |
| 0x5c | 0x44441111 | Character '~'. Lines 0-3. |
| 0x5d | 0x00000000 | Character '~'. Lines 4-7. |
| 0x5e | 0x00000000 | Special Character. Lines 0-3. |
| 0x5f | 0x00000000 | Special Character. Lines 4-7. |

To write the previous example data into the Font RAM and to program the OSD to use this data, the host processor must first select Font RAM 0 or 1 by writing to the *GC Write Bank Address* register (address offset 0x00A0). Once the Font RAM is selected, the host processor must write the data found in Table D-12 to the *GC Data* register (address offset 0x00A8). All data is written to the same OSD register address. Once all the font data is written, the host processor can enable the Font RAM by writing to the *GC Active Bank Address* register (address offset 0x00A4). This will cause the OSD to use the new font data during the next video frame.

Table D-13 shows the OSD register addresses and the data written by the host processor. This example assumes that the host is configuring Font RAM 1 of the Graphics Controller on layer 2.

*Table D-13:* **Example Color RAM Host Processor Writes**

| Address Offset | Data[31:0] | Description |
|---|---|---|
| 0x00A0 | 0x00000207 | Sets the Graphics Controller Number to 2 and selects Color RAM 1. |
| 0x00A8 | 0x00000000 | Character ' ' (space). Lines 0-3. |
| 0x00A8 | 0x00000000 | Character ' ' (space). Lines 4-7. |
| 0x00A8 | 0x18181800 | Character '!'. Lines 0-3. |
| 0x00A8 | 0x00180018 | Character '!'. Lines 4-7. |
| 0x00A8 | 0x66666600 | Character '"' (double-quotes). Lines 0-3. |
| 0x00A8 | 0x00000000 | Character '"' (double-quotes). Lines 4-7. |
| … | | |
| 0x00A8 | 0x6e663c00 | Character '@'. Lines 0-3. |
| 0x00A8 | 0x003e606e | Character '@'. Lines 4-7. |
| 0x00A8 | 0x663c1800 | Character 'A'. Lines 0-3. |
| 0x00A8 | 0x00667e66 | Character 'A'. Lines 4-7. |
| 0x00A8 | 0x7c667c00 | Character 'B'. Lines 0-3. |
| 0x00A8 | 0x007c6666 | Character 'B'. Lines 4-7. |
| … | | |
| 0x00A8 | 0x04101050 | Character '}'. Lines 0-3. |
| 0x00A8 | 0x00501010 | Character '}'. Lines 4-7. |
| 0x00A8 | 0x44441111 | Character '~'. Lines 0-3. |
| 0x00A8 | 0x00000000 | Character '~'. Lines 4-7. |
| 0x00A8 | 0x00000000 | Special Character. Lines 0-3. |
| 0x00A8 | 0x00000000 | Special Character. Lines 4-7. |
| 0x00A4 | 0x04000000 | Sets Color RAM 1 of Graphics controller 2 active. |

Host processor writes are shown for the General Purpose Processor Interface (little-endian).

Writing to the *GC Active Bank Address* register will affect all selected memories for all Graphics Controllers. A read-modify-write operation is best performed on this register to avoid incorrectly selecting an invalid memory.

The graphics controller can also be configured for an 8x8 2-bits per pixel font and a 16x16 1-bit per pixel font. Both of these modes are a 16-bit per line configuration with two lines of font data written to the Font RAM with every host processor write.

## Text RAM

The text RAM stores null terminated ASCII strings. The maximum number of strings the text RAM can store is configured by the "Number of Strings" parameter of the CORE Generator GUI and can be set to any value between 1 and 256. The maximum string length for each string is configured by the "Maximum String Length" parameter and can be set to 32, 64, 128 or 256. See Chapter 3, Customizing and Generating the Core for more information.

Table D-14 shows an example Text RAM and its contents with the "Number of Strings" parameter set to 4 and the "Maximum String Length" parameter set to 8. The four strings stored in the Text RAM in this example are "String0," "Text001," "STRING2" and "Xilinx3."

*Table D-14:* **Example Text RAM Memory Map**

| Address | Data[31:0] | Description |
|---|---|---|
| 0x00 | 0x69727453 | Start of String 0. Substring "Stri" |
| 0x01 | 0x0030676e | Continuation of String 0. Substring "ng0" |
| 0x02 | 0x74786554 | Start of String 1. Substring "Text" |
| 0x03 | 0x00313030 | Continuation of String 1. Substring "001" |
| 0x04 | 0x49525453 | Start of String 2. Substring "STRI" |
| 0x05 | 0x0032474e | Continuation of String 2. Substring "NG2" |
| 0x06 | 0x696c6958 | Start of String 3. Substring "Xili" |
| 0x07 | 0x0033786e | Continuation of String 3. Substring "nx3" |

Each text string must be terminated with a NULL character (0x00). If the string is not NULL character terminated, the text drawn could cause unpredictable results on screen. Any character after the first NULL (0x00) is ignored, and it does not matter what data is written after the first NULL character.

To write the previous example data into the Text RAM and to program the OSD to use this data, the host processor must first select Text RAM 0 or 1 by writing to the *GC Write Bank Address* register (address offset 0x00A0). Once the Text RAM is selected, the host processor must write the data found in Table D-14 to the *GC Data* register (address offset 0x00A8). All data is written to the same OSD register address. Once all the color data is written, the host processor can enable the Text RAM by writing to the *GC Active Bank Address* register (address offset 0x00A4). This will cause the OSD to use the new text data during the next video frame.

Table D-15 shows the OSD register addresses and the data written by the host processor. This example assumes that the host is configuring Text RAM 1 of the Graphics Controller on layer 4.

*Table D-15:* **Example Text RAM Host Processor Writes**

| Address Offset | Data[31:0] | Description |
|---|---|---|
| 0x00A0 | 0x00000405 | Sets the Graphics Controller Number to 4 and selects Text RAM 1. |
| 0x00A8 | 0x69727453 | Text data for Text RAM address 0x00 (String 0) |
| 0x00A8 | 0x0030676e | Text data for Text RAM address 0x01 (String 0) |
| 0x00A8 | 0x74786554 | Text data for Text RAM address 0x02 (String 1) |
| 0x00A8 | 0x00313030 | Text data for Text RAM address 0x03 (String 1) |
| 0x00A8 | 0x49525453 | Text data for Text RAM address 0x04 (String 2) |
| 0x00A8 | 0x0032474e | Text data for Text RAM address 0x05 (String 2) |
| 0x00A8 | 0x696c6958 | Text data for Text RAM address 0x06 (String 3) |
| 0x00A8 | 0x0033786e | Text data for Text RAM address 0x07 (String 3) |
| 0x00A4 | 0x00100000 | Sets Text RAM 1 of Graphics controller 4 active. |

Host processor writes are shown for the General Purpose Processor Interface (little-endian).

Writing to the *GC Active Bank Address* register will affect all selected memories for all Graphics Controllers. A read-modify-write operation is best performed on this register to avoid incorrectly selecting an invalid memory.

# EDK pCore Programmers Guide

This section introduces the concept of controlling the Xilinx Video On-Screen Display via a software driver. The EDK pCore address map and driver function calls are described.

## pCore Device Driver

The Xilinx On-Screen Display pCore includes a software driver written in the C Language that can be used to control the Xilinx OSD devices. A high-level API is provided and can be used without detailed knowledge of the Xilinx OSD devices. Application developers are encouraged to use this API to access the device features. A low-level API is also provided in case applications prefer to access the devices directly through the system registers described in the previous section.

Table D-16 lists the files that are included with the Xilinx OSD pCore driver and their description.

*Table D-16:* **Device Driver Source Files**

| File Name | Description |
|---|---|
| xosd.h | Contains all prototypes of high-level API to access all of the features of the Xilinx OSD devices. |
| xosd.c | Contains the implementation of high-level API to access all of the features of the Xilinx OSD devices except interrupts. |
| xosd_intr.c | Contains the implementation of high-level API to access interrupt feature of the Xilinx OSD devices. |
| xosd_sinit.c | Contains static initialization methods for the Xilinx OSD device driver. |
| xosd_g.c | Contains a template for configuration table of Xilinx OSD devices. This file is used by the high-level API and will be automatically generated to match the OSD device configurations by Xilinx EDK/SDK tools when the software project is built. |
| xosd_hw.h | Contains Low-level API (that is, register offset/bit definition and register-level driver API) that can be used to access the Xilinx OSD devices. |
| example.c | An example that demonstrates how to control the Xilinx OSD devices using the high-level API. |

# pCore API Functions

This section describes the functions included in the pcore Driver files generated for the Video On-Screen Display pCore. The software API is provide to allow easy access to the registers of the pCore as defined in Table 2-2 in the Register Space section. To utilize the API functions provided, the following header files must be included in the user's C code:

```
#include “xparameters.h”
#include “xosd.h”
```

The hardware settings of the system, including the base address of the Video On-Screen Display core are defined in the `xparameters.h` file. The `xosd.h` file provides the API access to all of the features of the Object Segmentation device driver.

More detailed documentation of the API functions can be found by opening the file `index.html` in the pCore directory `osd_v1_03_a/doc/html/api`.

## Functions in xosd.c

- int XOSD_CfgInitialize (XOSD *InstancePtr, XOSD_Config *CfgPtr, u32 EffectiveAddr)

  This function initializes an OSD device.

- void XOSD_SetBlankPolarity (XOSD *InstancePtr, int VerticalBlankPolarity, int HorizontalBlankPolarity)

  This function chooses the type of Vertical and Horizontal Blank Input Polarities.

- void XOSD_SetScreenSize (XOSD *InstancePtr, u32 Width, u32 Height)

  This function sets the screen size of the OSD Output.

- void XOSD_GetScreenSize (XOSD *InstancePtr, u32 *WidthPtr, u32 *HeightPtr)

This function gets the screen size of the OSD Output.

- void XOSD_SetBackgroundColor (XOSD *InstancePtr, u16 Red, u16 Blue, u16 Green)

  This function sets the Background color used by the OSD output.

- void XOSD_GetBackgroundColor (XOSD *InstancePtr, u16 *RedPtr, u16 *BluePtr, u16 *GreenPtr)

  This function gets the Background color used by the OSD output.

- void XOSD_SetLayerDimension (XOSD *InstancePtr, u8 LayerIndex, u16 XStart, u16 YStart, u16 XSize, u16 YSize)

  This function sets the start position and size of an OSD layer.

- void XOSD_GetLayerDimension (XOSD *InstancePtr, u8 LayerIndex, u16 *XStartPtr, u16 *YStartPtr, u16 *XSizePtr, u16 *YSizePtr)

  This function gets the start position and size of an OSD layer.

- void XOSD_SetLayerAlpha (XOSD *InstancePtr, u8 LayerIndex, u16 GlobalAlphaEnble, u16 GlobalAlphaValue)

  This function sets the Alpha value and mode of an OSD layer.

- void XOSD_GetLayerAlpha (XOSD *InstancePtr, u8 LayerIndex, u16 *GlobalAlphaEnblePtr, u16 *GlobalAlphaValuePtr)

  This function gets the Alpha value and mode of an OSD layer.

- void XOSD_SetLayerPriority (XOSD *InstancePtr, u8 LayerIndex, u8 Priority)

  This function sets the priority of an OSD layer.

- void XOSD_GetLayerPriority (XOSD *InstancePtr, u8 LayerIndex, u8 *PriorityPtr)

  This function gets the priority of an OSD layer.

- void XOSD_EnableLayer (XOSD *InstancePtr, u8 LayerIndex)

  This function enables an OSD layer.

- void XOSD_DisableLayer (XOSD *InstancePtr, u8 LayerIndex)

  This function disables an OSD layer.

- void XOSD_LoadColorLUTBank (XOSD *InstancePtr, u8 GcIndex, u8 BankIndex, u32 *ColorData)

  This function loads color LUT data into an OSD Graphics Controller Bank.

- void XOSD_LoadCharacterSetBank (XOSD *InstancePtr, u8 GcIndex, u8 BankIndex, u32 *CharSetData)

  This function loads Character Set data (font) into an OSD Graphics Controller Bank.

- void XOSD_LoadTextBank (XOSD *InstancePtr, u8 GcIndex, u8 BankIndex, u32 *TextData)

  This function loads Text data into an OSD Graphics Controller Bank.

- void XOSD_SetActiveBank (XOSD *InstancePtr, u8 GcIndex, u8 ColorBankIndex, u8 CharBankIndex, u8 TextBankIndex, u8 InstructionBankIndex)

  This function chooses active banks for a GC in an OSD device.

- void XOSD_CreateInstruction (XOSD *InstancePtr, u32 *InstructionPtr, u8 GcIndex, u16 ObjType, u8 ObjSize, u16 XStart, u16 YStart, u16 XEnd, u16 YEnd, u8 TextIndex, u8 ColorIndex)

This function creates an instruction for an OSD.

- void XOSD_LoadInstructionList (XOSD *InstancePtr, u8 GcIndex, u8 BankIndex, u32 *InstSetPtr, u32 InstNum)

  This function load an instruction list to be used by an Graphic Controller in an OSD device.

- void XOSD_GetVersion (XOSD *InstancePtr, u16 *Major, u16 *Minor, u16 *Revision)

  This function returns the version of an OSD device.

## Functions in xosd_sinit.c

- XOSD_Config * XOSD_LookupConfig (u16 DeviceId)

  XOSD_LookupConfig returns a reference to an XOSD_Config structure based on the unique device id, DeviceId.

## Functions in xosd_intr.c

- void XOSD_IntrHandler (void *InstancePtr)

  This function is the interrupt handler for the On-Screen-Display driver.

- int XOSD_SetCallBack (XOSD *InstancePtr, u32 HandlerType, void *CallBackFunc, void *CallBackRef)

  This routine installs an asynchronous callback function for the given HandlerType.

XILINX®

*Appendix E*

# C Model Reference

The Xilinx LogiCORE™ IP Video OSD has a bit accurate C model for 32-bit Windows, 64-bit Windows, 32-bit Linux and 64-bit Linux platforms. The model has an interface consisting of a set of C functions, which reside in a statically link library (shared library). Full details of the interface are given in Interface, page 115. An example piece of C code is provided in Example Code, page 125 to show how to call the model.

The model is bit accurate, as it produces exactly the same output data as the core on a frame-by-frame basis. However, the model is not cycle accurate, as it does not model the core's latency or its interface signals. The latest version of the model is available for download on the Xilinx LogiCORE IP Video OSD web page at:

http://www.xilinx.com/products/ipcenter/EF-DI-OSD.htm

## Unpacking and Model Contents

Unzip the `v_osd_v3_0_bitacc_model.zip` file, containing the bit accurate models for the On-Screen Display IP Core. This creates the directory structure and files in Table E-1.

*Table E-1:* **Directory Structure and Files of the Video On-Screen Display v3.0 Bit Accurate C Model**

| File Name | Contents |
|---|---|
| README.txt | Release notes |
| pg010_v_osd.pdf | *LogiCORE IP Video On-Screen Display Product Guide* |
| Makefile | Makefile for running GCC via make for 32-bit and 64-bit Linux platforms |
| v_osd_v3_0_bitacc_cmodel.h | Model header file |
| rgb_utils.h | Header file declaring the RGB image/video container type and support functions |
| yuv_utils.h | Header file declaring the YUV (.yuv) image file I/O functions |
| bmp_utils.h | Header file declaring the bitmap (.bmp) image file I/O functions |
| video_utils.h | Header file declaring the generalized image/video container type, I/O and support functions |
| video_fio.h | Header file declaring support functions for test bench stimulus file I/O |
| run_bitacc_cmodel.c | Example code calling the C model |

*Table E-1:* **Directory Structure and Files of the Video On-Screen Display v3.0 Bit Accurate C Model *(Cont'd)***

| File Name | Contents |
|---|---|
| run_bitacc_cmodel_config.c | Example code calling the C model; uses command line and config file arguments |
| /lin64 | Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms |
| libIp_v_osd_v3_0_bitacc_cmodel.so | Model shared object library |
| libstlport.so.5.1 | STL library, referenced by libIp_ v_osd_v3_0_bitacc_cmodel.so |
| run_bitacc_cmodel | 64-bit Linux fixed configuration executable |
| run_bitacc_cmodel_config | 64-bit Linux programmable configuration executable |
| /lin | Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms. |
| libIp_v_osd_v3_0_bitacc_cmodel.so | Model shared object library |
| libstlport.so.5.1 | STL library, referenced by libIp_ v_osd_v3_0_bitacc_cmodel.so |
| run_bitacc_cmodel | 32-bit Linux fixed configuration executable |
| run_bitacc_cmodel_config | 32-bit Linux programmable configuration executable |
| /nt64 | Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms |
| libIp_v_osd_v3_0_bitacc_cmodel.lib | Precompiled library file for 64-bit Windows platforms compilation |
| run_bitacc_cmodel.exe | 64-bit Windows fixed configuration executable |
| run_bitacc_cmodel_config.exe | 64-bit Windows programmable configuration executable |
| /nt | Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms |
| libIp_v_osd_v3_0_bitacc_cmodel.lib | Precompiled library file for 32-bit Windows platforms compilation |
| run_bitacc_cmodel.exe | 32-bit Windows fixed configuration executable |
| run_bitacc_cmodel_config.exe | 32-bit Windows programmable configuration executable |
| examples | Example input files to be used with the run_bitacc_cmodel_config executable |
| example0.cfg | Example config file; internal test patterns, no graphics controller and BMP output |
| example1.cfg | Example config file; no input, internal test patterns , no graphics controller and YUV output |

*Table E-1:* **Directory Structure and Files of the Video On-Screen Display v3.0 Bit Accurate C Model *(Cont'd)***

| File Name | Contents |
|---|---|
| example2.cfg | Example config file; BMP input, no graphics controller and BMP output |
| example3.cfg | Example config file., BMP input, graphics overlay and BMP output |
| clut.txt | Example graphics controller color look-up table/ pallet file |
| string.txt | Example graphics controller text/strings file |
| font.txt | Example graphics controller font file |
| instructions.txt | Example graphics controller instruction list |
| bridge.bmp | Example 24-bit 576x720 bitmap |

# Installation

For Linux, make sure the following files are in a directory in the $LD_LIBRARY_PATH environment variable:

- `libIp_v_osd_v3_0_bitacc_cmodel.so`
- `libstlport.so.5.1`

# Software Requirements

The Video On-Screen Display C models were compiled and tested with the software listed in Table E-2.

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

| Platform | C Compiler |
|---|---|
| 64-bit Linux | GCC 3.4.6 & 4.1.1 |
| 32-bit Linux | GCC 3.4.6 & 4.1.1 |
| 64-bit Windows | Microsoft Visual Studio 2005 |
| 32-bit Windows | Microsoft Visual Studio 2005 |

# Interface

The video OSD bit accurate C model core function is a statically linked library. This model is accessed through a set of functions and data structures that are declared in the `v_osd_v3_0_bitacc_cmodel.h` file. A higher level software project can make function calls to this function:

```
/**
 * Create a new state structure for this C-Model.
 *
 * IMPORTANT: Client is responsible for calling
 *            xilinx_ip_v_osd_v3_0_destroy_state()
 *            to free state memory.
```

```
 *
 * @param generics    Generics to be used to configure C-Model
 *                    state.
 *
 * @returns xilinx_ip_v_osd_v3_0_state*  Pointer to the internal
 *                                       state.
 */
struct xilinx_ip_v_osd_v3_0_state*
xilinx_ip_v_osd_v3_0_create_state(struct xilinx_ip_v_osd_v3_0_generics generics);

/**
 * Simulate this bit-accurate C-Model.
 *
 * @param     state      Internal state of this C-Model. State
 *                       may span multiple simulations.
 * @param     inputs     Inputs to this C-Model.
 * @param     outputs    Outputs from this C-Model.
 *
 * @returns   Exit code   Zero for SUCCESS, Non-zero otherwise.
 */
int xilinx_ip_v_osd_v3_0_bitacc_simulate
(
 struct xilinx_ip_v_osd_v3_0_state*   state,
 struct xilinx_ip_v_osd_v3_0_inputs   inputs,
 struct xilinx_ip_v_osd_v3_0_outputs* outputs
);
```

Before using the model, the structures holding the generics, inputs, and outputs of the OSD instance must be defined:

```
struct  xilinx_ip_v_osd_v3_0_generics generics;
struct  xilinx_ip_v_osd_v3_0_inputs   inputs;
struct  xilinx_ip_v_osd_v3_0_outputs  outputs;
```

The declaration of these structures is in the `v_osd_v3_0_bitacc_cmodel.h` file.

Before making the function call, complete these steps:

1. Populate the *generics* structure. It defines the values of build time parameters. See OSD Generics Structure for more information on the structure and an example of how to initialize.
2. Populate the *inputs* structure. It defines the values of run time parameters. See OSD Inputs Structure for more information on the structure and an example of how to initialize.
3. Populate the *outputs* structure. See OSD Outputs Structure for more information on the structure and an example of how to initialize.

After the inputs are defined and all `video_structs` are initialized, the model can be simulated by calling the following functions:

```
state = xilinx_ip_v_osd_v3_0_create_state(generics);
if (state == NULL) {
  printf("ERROR: could not create state object\n");
  return 1;
}

// Simulate the core
printf("Running the C model...\n");
if(xilinx_ip_v_osd_v3_0_bitacc_simulate(state, inputs, &outputs) != 0) {
  printf("ERROR: simulation did not complete successfully\n");
```

```
  return 1;
} else {
  printf("Simulation completed successfully\n");
}
```

The results are provided in the outputs structure, which contains only one member of type `video_struct`. See OSD Video Structure for more information on `video_struct`.

The successful execution of all provided functions return a value of 0, otherwise a non-zero error code indicates that problems occurred during function calls.

## OSD Generics Structure

The Xilinx LogiCORE IP Video OSD Core bit accurate C model takes multiple generic parameters. All generic parameters are integers or integer arrays. See Table E-3 for generic definitions.

*Table E-3:* **OSD Generics Structure**

| Generic | Designation |
|---|---|
| C_DATA_WIDTH | Data width of each color component channel; valid values are 8, 10 and 12. |
| C_NUM_LAYERS | The number of layers. |
| C_LAYER_TYPE[8] | Defines the layer type of each layer:<br>• 1=Graphics Controller<br>• 2=VFBC<br>• 3=XSVI<br>All other values are reserved. |
| C_LAYER_INS_BOX_EN[8] | Enable box instructions. |
| C_LAYER_INS_TEXT_EN[8] | Enable text instructions. |
| C_LAYER_CLUT_SIZE[8] | Maximum number of colors. |
| C_LAYER_TEXT_NUM_STRINGS[8] | Maximum number of strings. |
| C_LAYER_TEXT_MAX_STRING_LENGTH[8] | Maximum string length. |
| C_LAYER_FONT_NUM_CHARS[8] | Maximum number of characters. |
| C_LAYER_FONT_WIDTH[8] | Maximum font width. |
| C_LAYER_FONT_HEIGHT[8] | Maximum font height. |
| C_LAYER_FONT_BPP[8] | Font bits per pixel. |
| C_LAYER_FONT_ASCII_OFFSET[8] | The ASCII value of the first character in the font file. |

Calling `xilinx_ip_v_osd_v3_0_get_default_generics()` initializes the generics structure, `xilinx_ip_v_osd_v3_0_generics`, with the OSD defaults. An example of initialization of the generics structure with layer two configured as a graphics controller is as follows:

```
generics = xilinx_ip_v_osd_v3_0_get_default_generics(); //Get Defaults
generics.C_NUM_LAYERS = 3;
generics.C_LAYER_TYPE[2] = 1; // Graphics Controller
```

```
// Setup Graphics Controller
generics.C_LAYER_INS_BOX_EN[2]  = 1;
generics.C_LAYER_INS_TEXT_EN[2] = 1;
generics.C_LAYER_CLUT_SIZE[2]   = 256;

// Setup Font RAM
generics.C_LAYER_FONT_NUM_CHARS[2]    = 128;
generics.C_LAYER_FONT_WIDTH[2]        = 8;
generics.C_LAYER_FONT_HEIGHT[2]       = 8;
generics.C_LAYER_FONT_BPP[2]          = 1;
generics.C_LAYER_FONT_ASCII_OFFSET[2] = 0;

// Setup Text RAM
generics.C_LAYER_TEXT_NUM_STRINGS[2] = 16; // Set number of strings
generics.C_LAYER_TEXT_MAX_STRING_LENGTH[2] = 64; //Set max string length
```

## OSD Inputs Structure

The structure `xilinx_ip_v_osd_v3_0_inputs` defines the values of run time parameters and the actual input video frames/images for each layer.

```
struct xilinx_ip_v_osd_v3_0_inputs
{
  struct  video_struct video_in[OSD_MAX_LAYERS];

  struct frame_cfg_struct * frame_cfg;
  struct layer_cfg_struct *layer_cfg[OSD_MAX_LAYERS];
  struct graphics_cfg_struct * gfx_cfg[OSD_MAX_LAYERS];

  int     num_frames;
  int     color_space;

}; // end xilinx_ip_v_osd_v3_0_inputs
```

The `video_in` variable is an array of `video_struct` structures, one structure per layer. See the OSD Video Structure for a description of the `video_in` structure. The `video_in` structure must be initialized if neither the internal graphics controller nor the test pattern generator is used.

### Frame Configuration

The `frame_cfg` variable is a pointer to the `frame_cfg_struct`. The `frame_cfg_struct` is defined as:

```
struct frame_cfg_struct
{
  int y_size;
  int x_size;
  int bg_color[3];

  struct frame_cfg_struct * next; // For Changing parameters each Frame
};
```

The `frame_cfg` variable points to the first element in the frame config linked list. For each frame, the OSD model reads the x and y size of output frame and the background color from the `frame_cfg_struct` pointed to by `frame_cfg`. At the end of the frame, if the next pointer is not NULL, the OSD model updates the background color and the output size from the next structure in the linked list. Consequently, if the number of video frames

is more than the number of elements in the linked list, the last element is used for the remaining frames. The user is responsible for initializing the linked list.

## Layer Configuration

The `layer_cfg` variable is an array of pointers to the `layer_cfg_struct` structure, one pointer per layer. The `layer_cfg_struct` is defined as:

```
struct layer_cfg_struct
{
  int enable;
  int g_alpha_en;
  int priority;
  int alpha;
  int x_pos;
  int y_pos;
  int x_size;
  int y_size;

  int chan_mode[4];
  int chan_color[4];

  struct layer_cfg_struct * next; // For Changing parameters each Frame
};
```

Each pointer must be initialized to point to the first element in the layer config linked list. For each frame, the OSD model reads the layer registers and the test parameter arrays (`chan_mode[4]` and `chan_color[4]`) from the `layer_cfg_struct` pointed to by the `layer_cfg` pointer. This linked list enables the user to change the layer configuration (size, position, transparency, z-plane, and so on) for each video frame.

At the end of the frame, if the next pointer is not NULL, the OSD model updates the layer configuration from the next structure in the linked list. Consequently, if the number of video frames is more than the number of elements in the linked list, the last element is used for the remaining frames. The user is responsible for initializing the linked list.

## Graphics Configuration

The `gfx_cfg` variable is an array of pointers to the `graphics_cfg_struct` structure, one pointer per layer. This variable is only used if the layer is configured for graphics controller input. The `graphics_cfg_struct` is defined as:

```
struct graphics_cfg_struct
{
  int layer_num;

  uint16 * clut; // Color Table
  char * text_ram; // Text Ram
  int * font_ram; // Font Ram

  struct graphics_list * graph_instruction;

  struct graphics_cfg_struct * next; // For Changing parameters each
Frame
};
```

Each pointer must be initialized to point to the first element in the graphics config linked list. For each frame, the OSD model reads the graphics controller memories from the `graphics_cfg_struct` pointed to by the `gfx_cfg` pointer. This linked list enables the

user to change the graphics controller output (boxes, text, color, size, position, transparency, font and strings) for each video frame.

The CLUT pointer points to an array of 16-bit unsigned integers. This array contains the color entries for the current video frame. Each color entry contains four integers, one for each color component and one for alpha. The CLUT array must contain 4*16 or 4*256 integers.

The `text_ram` pointer points to an array of characters. This array contains all strings for the current video frame. The number of characters in the array must equal the (maximum string length) * (the number of strings).

The `font_ram` pointer points to an array of integers. This array contains the font for the current video frame. The number of integers in the array must equal the (number of characters) * (font width) * (font height). The number of bits used in each integer is 8, 16 or 32 depending on the setting of the `font_width` and `font_bpp`.

The `graph_instruction` pointer points to a linked list of graphics instructions (defined by the `graphics_list` structure). This linked list contains the graphics instructions for the current video frame. The Graphics Controller draws each instruction in the linked list until a NULL pointer is encountered. The `graphic_list` structure is defined as:

```
struct graphics_list
{
  int opcode;
  int xstart;
  int xstop;
  int ystart;
  int ystop;
  int color_index;
  int text_index;
  int object_size;

  struct graphics_list * next;

};
```

This structure contains the same fields as in the instruction file defined previously. The opcode variable can be `OSD_INS_BOX`, `OSD_INS_TEXT` or `OSD_INS_BOXTEXT` (each defined in the `v_osd_v_2_0_bitacc_cmodel.h` file). See Table D-1, page 92 through Table D-5, page 95 for more information on xstart, xstop, ystart, ystop, color_index and text_index definitions.

At the end of the frame, if the next pointer is not NULL, the OSD model updates the graphics controller configuration from the next structure in the linked list. Consequently, if the number of video frames is more than the number of elements in the linked list, the last element is used for the remaining frames. The user is responsible for initializing the linked list. Example initialization code of the inputs structure is as follows:

```
  inputs.frame_cfg   = (struct frame_cfg_struct *) calloc(1,
sizeof(struct frame_cfg_struct));
  inputs.frame_cfg->x_size      = 1280;
  inputs.frame_cfg->y_size      =  720;
  inputs.frame_cfg->bg_color[0] = 0x88;
  inputs.frame_cfg->bg_color[1] = 0x3a;
  inputs.frame_cfg->bg_color[2] = 0xbd;
  inputs.frame_cfg->next        = NULL; // End of Frame Config

  // Setup Layer 0 Configuration
```

```
   inputs.layer_cfg[0] = (struct layer_cfg_struct *) calloc(1,
sizeof(struct layer_cfg_struct));
  inputs.layer_cfg[0]->enable      = 1;
  inputs.layer_cfg[0]->g_alpha_en  = 0;
  inputs.layer_cfg[0]->priority    = 2;
  inputs.layer_cfg[0]->alpha       = 0x80;
  inputs.layer_cfg[0]->x_pos       = 0;
  inputs.layer_cfg[0]->y_pos       = 0;
  inputs.layer_cfg[0]->x_size      = 1280;
  inputs.layer_cfg[0]->y_size      = 720;

  inputs.layer_cfg[0]->chan_mode[0] = OSD_SOLID_MODE;
  inputs.layer_cfg[0]->chan_mode[1] = OSD_SOLID_MODE;
  inputs.layer_cfg[0]->chan_mode[2] = OSD_SOLID_MODE;
  inputs.layer_cfg[0]->chan_mode[3] = OSD_HRAMP_MODE;

  inputs.layer_cfg[0]->chan_color[0] = 0xe0;
  inputs.layer_cfg[0]->chan_color[1] = 0x5a;
  inputs.layer_cfg[0]->chan_color[2] = 0xbf;
  inputs.layer_cfg[0]->chan_color[3] = 0x80; // Alpha
  inputs.layer_cfg[0]->next = NULL;
```

## OSD Outputs Structure

The structure `xilinx_ip_v_osd_v3_0_outputs` provides the actual output video frames/images of the OSD core. This structure is a wrapper to the standard `video_struct` used by other Xilinx video core C models.

```
struct xilinx_ip_v_osd_v3_0_outputs
{
  struct  video_struct video_out;
}; // xilinx_ip_v_osd_v3_0_outputs
```

The `video_out` structure must be initialized. The following code shows a typical `video_out` initialization.

```
  // Setup Output Video Buffer
  outputs.video_out.frames          = inputs.num_frames;
  outputs.video_out.rows            = inputs.frame_cfg->y_size;
  outputs.video_out.cols            = inputs.frame_cfg->x_size;
  outputs.video_out.mode            = FORMAT_C444;
  outputs.video_out.bits_per_component = generics.C_DATA_WIDTH;
  outputs.video_out.data[0]         = NULL;
  outputs.video_out.data[1]         = NULL;
  outputs.video_out.data[2]         = NULL;
```

## OSD Video Structure

Input images or video streams can be provided to the OSD v3.0 reference model using the `video_struct` structure, defined in `video_utils.h`. Output images or video streams are also placed within a `video_struct` structure. The video_struct is defined as:

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

The structure member variables are defined in Table E-4.

*Table E-4:* **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<br>Pertains to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through the all frames of the video stream, however different planes, such as y, u and v can have different smaller dimensions. |
| cols | Number of columns per frame<br>Pertains to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through the all frames of the video stream, however different planes, such as y, u and v can have different smaller dimensions. |
| bits_per_component | Number of bits per color channel/component.<br>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.<br>Named constants to be assigned to `mode` are listed in Table E-5. |
| data | Set of 5 pointers to 3 dimensional arrays containing data for image planes.<br>`data` is in 16 bit unsigned integer format accessed as data[plane][frame][row][col] |

Add Note that the following formats are supported by the OSD.

***Note:*** The OSD core supports four formats: FORMAT_RGB, FORMAT_C444, FORMAT_C422, and FORMAT_C420

*Table E-5:* **Named Constants for Video Modes With Corresponding Planes and Representations**

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

## Working With Video_struct Containers

The `video_utils.h` 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);
```

Function `video_planes_per_mode` returns the number of component planes defined by the mode variable, as described in Table E-5. Functions `video_rows_per_plane` and `video_cols_per_plane` 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 variable `in_video`, with this construct:

```
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]
      }
    }
  }
}
```

### Delete the Video Structure

Finally, large arrays such as the `video_in` element in the video structure must be deleted to free up memory. As an example, the following function is defined as part of the `video_utils` package.

```
void free_video_buff(struct video_struct* video )
{
  int plane, frame, row;

  if (video->data[0] != NULL) {
    for (plane = 0; plane <video_planes_per_mode(video->mode); plane++) {
      for (frame = 0; frame < video->frames; frame++) {
        for (row = 0; row<video_rows_per_plane(video,plane); row++) {
          free(video->data[plane][frame][row]);
        }
        free(video->data[plane][frame]);
      }
      free(video->data[plane]);
    }
  }
}
```

This function can be called in the following way to free the video input buffers (up to eight) and the video output buffer:

```
// Free Layer Buffers
for(i=0; i < generics.C_NUM_LAYERS; i++)
{
  printf("Freeing Layer Video Buffer #%d...\n", i);
  free_video_buff(&inputs.video_in[i]);
}
printf("Freeing Output Buffer...\n");
free_video_buff(&outputs.video_out);
```

## Example Code

Two example C files, `run_bitacc_cmodel.c` and `bitacc_cmodel_config.c`,are provided. The 32-bit and 64-bit Windows and Linux executables for these examples are also included. This C file has these characteristics:

The `run_bitacc_cmodel` example executable provides:

- Shows a fixed implementation of the OSD, including two VFBC layers populated from the internal test pattern generator and one graphics controller layer.
- Contains an example of how to write an application that makes all necessary function calls to the OSD C model core function.
- Contains an example of how to populate the video structures at the input and output, including allocation of memory to these structures.
- Uses a YUV file reading function to extract video information from YUV files for use by the model.
- Uses a YUV file writing function to provide an output YUV file, which allows the user to visualize the result of the core.

The `run_bitacc_cmodel` example executable does not use command line parameters. To run the executable:

1. Use the **`cd`** command to go to the platform directory (lin64, lin, win64 or win32).
2. Enter this command at the shell or DOS prompt:

```
run_bitacc_cmodel
```

The `run_bitacc_cmodel_config` example executable provides:

- Shows configurable implementations of the OSD configured from a config file or command line arguments.
- Includes a config file parser, allowing the user to pass parameters into the model for multiple test cases.
- Uses YUV or BMP file reading functions to extract video information from YUV or BMP files for use by the model.
- Uses YUV or BMP file writing functions to provide an output YUV or BMP file, which allows the user to visualize the result of the core.

The `run_bitacc_cmodel` _config example executable uses multiple command line parameters. To run the executable:

1. Use the **`cd`** command to go to the platform directory (lin64, lin, win64 or win32).
2. Enter this command at the shell or DOS prompt:

**`run_bitacc_cmodel_config -i <`***Config Filename***`> <`***-parameter=value …***`>`**

## Config File Format

The config file defines configuration generics, register settings and test parameters for each video frame to be simulated by the C model. The basic file format is a series of lines each containing a parameter-value pair separated by an '='. An example config file snippet is provided here:

```
C_DATA_WIDTH = 8
C_NUM_LAYERS = 2
T_NUM_FRAMES = 2
# FORMAT_RGB
T_COLORSPACE = 8
C_NUM_DATA_CHANNELS = 3
C_OUTPUT_MODE = 1
C_LAYER0_TYPE = 2
C_LAYER1_TYPE = 2
T_OUTFILE = example0.bmp

[FRAME 1]
R_X_SIZE = 1280
R_Y_SIZE = 720
R_BGCOLOR0 = 0x10
R_BGCOLOR1 = 0x80
R_BGCOLOR2 = 0x80

R_LAYER0_ENABLE = 1
R_LAYER0_G_ALPHA_EN = 1
R_LAYER0_PRIORITY = 1
R_LAYER0_ALPHA = 0xff
R_LAYER0_X_POS = 0
R_LAYER0_Y_POS = 0
```

```
R_LAYER0_X_SIZE = 640
R_LAYER0_Y_SIZE = 720

T_LAYER0_CHAN0_MODE = 5
T_LAYER0_CHAN1_MODE = 5
T_LAYER0_CHAN2_MODE = 5
T_LAYER0_CHAN3_MODE = 5
T_LAYER0_CHAN0_COLOR = 2
T_LAYER0_CHAN1_COLOR = 0xa0
T_LAYER0_CHAN2_COLOR = 0xb0
T_LAYER0_CHAN3_COLOR = 0xc0
```

Configuration generics are prefixed with “C_”, OSD hardware registers are prefixed with “R_” and test parameters are prefixed with “T_”. Settings can be changed for each video frame. Video frame settings are delineated by a single line containing “[FRAME <*num*>]”, where <*num*> is an integer denoting the frame number. Global parameters (generics and some test parameters) must be before the first “[FRAME <*num*>]” line. Comment lines are those lines in which the first non-white-space character is '#' or ';'. See Table E-6 for a full list of all valid parameters.

*Table E-6:* **Global Parameters**

| Parameter | Valid Range | Description |
|---|---|---|
| Global Parameters | | Global parameters must be outside of [FRAME <*num*>] sections. |
| C_DATA_WIDTH | 8,10,12 | Data width of each color component channel. |
| C_NUM_LAYERS | 1-8 | Number of layers. |
| C_LAYER<*num*>_TYPE | 1,2,3 | Defines the layer type:<br>1 = Graphics Controller<br>2 = VFBC. Loads data from a file or from an internally generated test pattern. The T_LAYER<*num*>_CHAN0_MODE (see below) defines if the layer data is from internal test pattern or from file. If the T_COLORSPACE is set to 8, the file format expected is .bmp. If T_COLOR_SPACE is set to 1,2 or 3, the file format expected is .yuv.<br>3 = XSVI. Same as VFBC. |
| C_LAYER<*num*>_INS_BOX_EN | 0,1 | Enable Box instructions. If 0, then all box instructions in the instruction files are ignored. |
| C_LAYER<*num*>_INS_TEXT_EN | 0,1 | Enable Text Instructions. If 0, then all text instructions in the instruction files are ignored.<br>Both C_LAYER<*num*>_INS_BOX_EN and C_LAYER<*num*>_INS_TEXT_EN must be enabled to enable the box text instruction. |
| C_LAYER<*num*>_IMEM_SIZE | 4-4096 | Maximum number of instructions . |
| C_LAYER<*num*>_CLUT_SIZE | 16 or 256 | Maximum number of colors. |
| C_LAYER<*num*>_TEXT_NUM_STRINGS | 1 – 256 | Maximum number of strings. |
| C_LAYER<*num*>_TEXT_MAX_STRING_LENGTH | 32,64,128,256 | Maximum string length. |

*Table E-6:* **Global Parameters *(Cont'd)***

| Parameter | Valid Range | Description |
|---|---|---|
| C_LAYER<*num*>_FONT_NUM_CHARS | 1-256 | Maximum number of characters. |
| C_LAYER<*num*>_FONT_WIDTH | 8,16 | Maximum Font Width. |
| C_LAYER<*num*>_FONT_HEIGHT | 8,16 | Maximum Font Height. |
| C_LAYER<*num*>_FONT_BPP | 1,2 | Font bits per pixel. 1 corresponds to 2 color font and 2 corresponds to 4 color font. |
| C_LAYER<*num*>_FONT_ASCII_OFFSET | 0 - (C_LAYER<*num*>_FONT_NUM_CHARS) -1 | ASCII value of the first character in the font file. |
| T_NUM_FRAMES | 1- | Number of frames to simulate |
| T_COLORSPACE | 1,2,3,8 | Color space:<br>1 = YUV 4:2:0<br>2 = YUV 4:2:2<br>3 = YUV 4:4:4<br>8 = RGB |
| T_OUTFILE | Any String | Destination file name to write output data. If the T_COLORSPACE is set to 8, this file will be in 24-bit .bmp format, otherwise this file is a planar .yuv file. |
| T_LAYER<*num*>_VIDEO_FILE | Any String | Defines the .bmp or .yuv file used to read layer data if the C_LAYER<*num*>_TYPE is set to 2. |
| T_LAYER<*num*>_INSTRUCTION_FILE | Any String | File name of instruction file. The OSD C model does not include a default set of instructions internally. This parameter must be set if using the graphics controller. See Instruction File Format. |
| T_LAYER<*num*>_CLUT_FILE | Any String | File name of color LUT file. The OSD C model does not include a default color LUT internally. This parameter must be set if using the graphics controller. See Color LUT File Format. |
| T_LAYER<*num*>_FONT_FILE | Any String | File name of font file. The OSD C model does not include a default font internally. This parameter must be set if using the graphics controller. See Font File Format. |
| T_LAYER<*num*>_TEXT_FILE | Any String | File name of string file. The OSD C model does not include a default set of strings internally. This parameter must be set if using the graphics controller. See String File Format. |
| Frame Parameters | | Frame Parameters can be defined and redefined for each frame. |
| R_X_SIZE | 1 – 4096 | Width of OSD output frames. |
| R_Y_SIZE | 1 – 4096 | Height of OSD output frames. |

*Table E-6:* **Global Parameters *(Cont'd)***

| Parameter | Valid Range | Description |
|---|---|---|
| R_BGCOLOR0 | 0x00 – 0xfff | Background color component 0 – R or Y.<br>Maximum value for data width of 8 is 0xff.<br>Maximum value for data width of 10 is 0x3ff.<br>Maximum value for data width of 12 is 0xfff. |
| R_BGCOLOR1 | 0x00 – 0xfff | Background color component 1 – G or U.<br>Maximum value for data width of 8 is 0xff.<br>Maximum value for data width of 10 is 0x3ff.<br>Maximum value for data width of 12 is 0xfff. |
| R_BGCOLOR2 | 0x00 – 0xfff | Background color component 2 – B or V.Maximum value for data width of 8 is 0xff.<br>Maximum value for data width of 10 is 0x3ff.<br>Maximum value for data width of 12 is 0xfff. |
| R_LAYER<*num*>_ENABLE | 0,1 | Enables layer when 1. |
| R_LAYER<*num*>_G_ALPHA_EN | 0,1 | Enables global alpha when 1. When 0, pixel alpha values are used. |
| R_LAYER<*num*>_PRIORITY | 0-7 | Z-plane order. Lower values denotes layers that are below layers with higher priority. |
| R_LAYER<*num*>_ALPHA | 0-0xfff | Alpha value for 100% opaque to 100% transparent.<br>Maximum value for data width of 8 is 0xff.<br>Maximum value for data width of 10 is 0x3ff.<br>Maximum value for data width of 12 is 0xfff. |
| R_LAYER<*num*>_X_POS | 0 – (R_X_SIZE-1) | X position of upper-left corner of layer. |
| R_LAYER<*num*>_Y_POS | 0 – (R_Y_SIZE-1) | Y position of upper-left corner of layer. |
| R_LAYER<*num*>_X_SIZE | 0 – R_X_SIZE | Width of layer. |
| R_LAYER<*num*>_Y_SIZE | 0 – R_Y_SIZE | Height of layer. |

*Table E-6:* **Global Parameters *(Cont'd)***

| Parameter | Valid Range | Description |
|---|---|---|
| T_LAYER<*num*>_CHAN0_MODE | 0 – 7 | The test mode of color channel/component 0 (R or Y)<br>0 = OSD_PREFILL_MODE: Denotes that the layer buffer is pre-filled with data before the OSD core simulation begins. The OSD model will expect to read input data from the T_LAYER<*num*>_VIDEO_FILE in this mode.<br>1 = OSD_GRAPHICS_MODE: Denotes that the layer data will be generated from the graphics controller. All the graphics controller files must be setup.<br>2 = OSD_CHECKER_MODE: Channel data is generated from internal test pattern generator. Channel data filled with T_LAYER<*num*>_CHAN0_COLOR in the upper-left and lower-right quadrants and filled with the bit-reversed color in the upper-right and lower-left quadrants.<br>3 = OSD_RAND_MODE: Channel data is generated from internal test pattern generator. Channel data is filled with random data. The value of T_LAYER<*num*>_CHAN0_MODE is used as the seed.<br>4 = OSD_SOLID_MODE: Channel data is generated from internal test pattern generator. Channel data is filled with the value of T_LAYER<*num*>_CHAN0_MODE.<br>5 = OSD_HRAMP_MODE: Channel data is generated from internal test pattern generator. Channel data is filled with a horizontal ramp, values incremented every pixel.<br>6 = OSD_VRAMP_MODE: Channel data is generated from internal test pattern generator. Channel data is filled with a vertical ramp, values incremented every line.<br>7 = OSD_TEMPR_MODE: Channel data is generated from internal test pattern generator. Channel data is filled with a temporal ramp, values incremented every frame.<br>NOTE: If T_LAYER<*num*>_CHAN0_MODE is set to 0 or 1, then T_LAYER<*num*>_CHAN1_MODE through T_LAYER<*num*>_CHAN3_MODE is ignored. |
| T_LAYER<*num*>_CHAN1_MODE | 0 - 7 | Same as T_LAYER<*num*>_CHAN0_MODE for channel 1 |
| T_LAYER<*num*>_CHAN2_MODE | 0 - 7 | Same as T_LAYER<*num*>_CHAN0_MODE for channel 2 |
| T_LAYER<*num*>_CHAN3_MODE | 0 - 7 | Same as T_LAYER<*num*>_CHAN0_MODE for channel 3 (alpha) |

*Table E-6:* **Global Parameters *(Cont'd)***

| Parameter | Valid Range | Description |
|---|---|---|
| T_LAYER<*num*>_CHAN0_COLOR | 0 – 0xfff | Used when T_LAYER<*num*>_CHAN0_MODE is set to 2-7. Used to set the color or to configure the internal test pattern generator for channel 0.<br>Maximum value for data width of 8 is 0xff.<br>Maximum value for data width of 10 is 0x3ff.<br>Maximum value for data width of 12 is 0xfff. |
| T_LAYER<*num*>_CHAN1_COLOR | 0 – 0xfff | Same as T_LAYER<*num*>_CHAN0_COLOR for channel 1<br>Maximum value for data width of 8 is 0xff.<br>Maximum value for data width of 10 is 0x3ff.<br>Maximum value for data width of 12 is 0xfff. |
| T_LAYER<*num*>_CHAN2_COLOR | 0 – 0xfff | Same as T_LAYER<*num*>_CHAN0_COLOR for channel 2<br>Maximum value for data width of 8 is 0xff.<br>Maximum value for data width of 10 is 0x3ff.<br>Maximum value for data width of 12 is 0xfff. |
| T_LAYER<*num*>_CHAN3_COLOR | 0 – 0xfff | Same as T_LAYER<*num*>_CHAN0_COLOR for channel 3 (alpha)<br>Maximum value for data width of 8 is 0xff.<br>Maximum value for data width of 10 is 0x3ff.<br>Maximum value for data width of 12 is 0xfff. |

## Color LUT File Format

The color LUT file defines the color pallet used by the graphics controller. Each graphics controller can have a different color LUT file just as the OSD hardware can have different color LUT memory. The format of the file is plain text containing a series of decimal or hexadecimal numbers separated by white space or new line characters. Only the lower 8-bits of each number are used.

The order of the file is channel0, channel1, channel2, and alpha for each color entry starting at entry zero. Here is an example color LUT file:

```
0x00 0x00 0x00 0x00
0x10 0x80  128 0xc0
0x51 0x5a 0xef 0x80
0x89 0x52 0x46  128
0x6b 0xba 0x65 0x80
```

The first line shows all color 0 and has all channels including alpha set to zero. The second line defines color 1 to be black in YUV with an alpha of 192. The remaining lines define color 2, 3 and 4 as red, green and blue in YUV, all with an alpha of 128 or 50% transparent.

The OSD can have a color LUT with 16 colors or 256 colors (64 or 1024 separate numbers for all channels). Not all entries need to be defined. Those entries not defined are set to zero. Consequently, the previous example defines only color entries 0, 1, 2, 3 and 4. Entries 5 through to the end of the table are zero.

Colors can be changed for each video frame (just as in the OSD hardware) by providing multiple color LUTs within the file. The first C_LAYER<*num*>_CLUT_SIZE numbers are

used for frame 1, the next C_LAYER<*num*>_CLUT_SIZE numbers are used for the next frame, and so on. If the number of frames is more than the number of color LUTs in the file, then the last color LUT is used for all remaining frames.

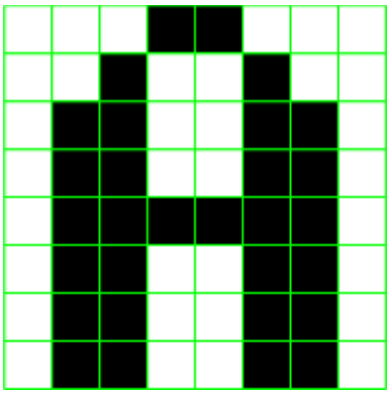
The Xilinx LogiCORE IP Video OSD C model does not include a default color LUT internally. The color LUT must be initialized from file if using the graphics controller.

## Font File Format

The font file defines the bits used to define each pixel of each line of each character used by the graphics controller. Each graphics controller can have a different font file just as the OSD hardware can have different font memory. The format of the file is plain text containing a series of decimal or hexadecimal numbers separated by white space or new-line characters.

The order of the file is line 0, line 1, line 2, etc for each character. The number of lines for each character is defined by the C_LAYER<num>_FONT_HEIGHT parameter. The number of bits for each line is defined by C_LAYER<*num*>_FONT_WIDTH * C_LAYER<*num*>_FONT_BPP. The first character in the font file does not have to define character 0. Instead, the first character is set by the C_LAYER<*num*>_FONT_ASCII_OFFSET. Here is an example font file:

0x18
0x24
0x66
0x66
0x7e
0x66
0x66
0x66



This example shows a snippet of the font file for C_LAYER<*num*>_FONT_WIDTH=8, C_LAYER<*num*>_FONT_HEIGHT=8, C_LAYER<*num*>_FONT_BPP=1 and C_LAYER<*num*>_FONT_ASCII_OFFSET=32. The eight lines shown are for the capital letter 'A', ASCII 65. These lines would be the 33rd (65-32) character definition and lines 265 through 272 in the font file.

Fonts can be changed in each video frame (just as in the OSD hardware) by providing multiple fonts within the file. The first C_LAYER<*num*>_FONT_NUM_CHARS * C_LAYER<*num*>_FONT_WIDTH * C_LAYER<*num*>_FONT_HEIGHT numbers are used for frame 1, the next C_LAYER<*num*>_FONT_NUM_CHARS *C_LAYER<*num*>_FONT_WIDTH * C_LAYER<*num*>_FONT_HEIGHT numbers are used for the next frame, and so on. If the number of frames is more than the number of fonts in the file, then the last font is used for all remaining frames.

The Xilinx LogiCORE IP Video OSD C model does not include a default font internally. The font must be initialized from a file if using the graphics controller.

## String File Format

The string file defines the text strings used by the graphics controller. Each graphics controller can have a different font file just as the OSD hardware can have different font memory. The format of the file is plain text containing one string of characters including spaces per line.

The order of the file is string 0, string 1, string 2, and so on, again, one string per line. The number of strings for each graphics controller is defined by this parameter:

C_LAYER<*num*>_TEXT_NUM_STRINGS

The maximum number of characters (including the terminating NULL character) is defined by this parameter:

C_LAYER<*num*>_TEXT_MAX_STRING_LENGTH parameter

Here is an example string file:

```
This is String # 0.  It is on one line!
String 1
Xilinx
OSD
Menu
!&^%!@#*
```

In the previous example file, only the first lines (up to C_LAYER<num>_TEXT_NUM _STRINGS number of lines) are used. All other lines are ignored. Also, the first characters of each line (up to C_LAYER<num>_TEXT_MAX_STRING_LENGTH) are used. All other characters are ignored. If the maximum string length was set to 8, the first string would be truncated to "This is\0".

***Note:*** In the OSD hardware, any character after the first NULL character in a string is ignored and not displayed.

Strings can be changed in each video frame by providing multiple sets of strings within the string file. The first C_LAYER<*num*>_TEXT_NUM_STRINGS number of lines are used for frame 1, the next C_LAYER<*num*>_TEXT_NUM_STRINGS number of lines are used for the next frame, and so on. If the number of frames is more than the number of sets of strings in the file, then the last set of strings are used for all remaining frames.

The Xilinx LogiCORE IP Video OSD C model does not include a default set of strings internally. The text strings must be initialized from a file if using the graphics controller.

## Instruction File Format

The instruction file defines the instructions used by the graphics controller. Each graphics controller can have a different instruction file just as the OSD hardware can have different instruction memory. The format of the file is plain text containing one string of characters including spaces per line. One full instruction is contained on each line.

The order of the file is instruction, x_start, x_stop, y_start, y_stop, color_index, text_index and object_size on each line. The instruction field is a text string describing the graphics instruction. All other fields are either decimal or hexadecimal numbers for the parameters of the instruction.

Here is an example instruction file:

```
############################
# Frame 1 Instructions
############################
BOX      10  20  40  80 1 0 4
```

```
BOX      40  60  70  90 3 0 4
TEXT    100 100  50  50 2 1 0x40
BOXTEXT  30  40  30  40 2 2 0x14
END
##########################
# Frame 2 Instructions
##########################
TEXT    100 100  50  50 2 1 0x40
BOX      20  40  40  80 1 0 4
BOX      40  80  70  90 3 0 4
TEXT    200 100  50  50 2 1 0x40
BOXTEXT  30  40  30  40 2 2 0x14
END
```

Each field is described in Table E-7.

*Table E-7:* **Instruction File Fields**

| Field | Valid Range | Description |
|---|---|---|
| Instruction | BOX, TEXT, BOXTEXT, END | The graphics instruction. |
| Xstart | 0 – end of line | Starting draw x position of the instruction. |
| Xstop | 0 – end of line | Ending draw x position of the instruction. |
| Ystart | 0 – end of frame | Starting draw y position of the instruction. |
| Ystop | 0 – end of frame | Ending draw y position of the instruction. |
| Color index | 0 – 15 or 255 | The color to be used for the graphics object.<br>For boxes, this color index is used directly.<br>For Text with BPP=1, the color index is used for the background and the color index + 1 is used for the foreground.<br>For Text with BPP=2, the color index is used for bits “00” in the font, color index + 1 for bits “01”, color index + 2 for “10” and color index + 3 for “11”. |
| Text index | 0 – (number of strings -1) | The text string to draw. |
| Object Size | 0 – 0xff | For BOX, Size of boxes.<br>For BOXTEXT, [3:0] size of boxes, [7;4] size of text.<br>For TEXT, bits [7:4] size of text. |

See Instruction RAM in Appendix D for more information on the format of each instruction.

There are two “END”s in the example instruction file because the file is used to describe the instructions for each video frame. All instructions from the beginning of the file to the first END are displayed on frame 1. For each frame following, the instructions between each subsequent “END” are displayed. If the number of frames is more than the number of “END”s in the file, then the last set of instructions are displayed for all remaining frames.

The Xilinx LogiCORE IP Video OSD C model does not include a default set of instructions internally. The instructions must be initialized from a file if using the graphics controller.

## Initializing the OSD 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`, `yuv_utils.h`, `rgb_utils.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 `rgb_utils.h` and `bmp_utils.h` files declare 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.

```
int write_bmp(FILE *outfile, struct rgb8_video_struct *rgb8_video);
int read_bmp(FILE *infile, struct rgb8_video_struct *rgb8_video);
```

These functions are used to dynamically allocate and free memory for RGB structure storage:

```
int alloc_rgb8_frame_buff(struct rgb8_video_struct* rgb8video );
void free_rgb_frame_buff(struct rgb_video_struct* rgb_video );
```

Exchanging data between rgb8_video_struct and general video_struct type frames/videos is facilitated by 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. Additionally, the input structure must have the dynamically allocated containers (data, r, g, b, y, u, and v arrays) 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.

### YUV Image/Video Files

The `yuv_utils.h` file declares functions that support file access in YUV format. These functions are used to dynamically allocate and free memory for YUV structure storage:

```
int alloc_yuv8_frame_buff(struct yuv8_video_struct* yuv8video );
void free_yuv_frame_buff(struct yuv_video_struct* yuv_video );
```

These functions allow reading and writing of YUV functions (used to initialize or write yuv8_video data):

```
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 functions:

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

YUV formats (4:2:0, 4:2:2 and 4:4:4) can be converted with these functions:

```
int yuv8_420to444(struct yuv8_video_struct* video_in, struct yuv8_video_struct* video_out);
int yuv8_422to444(struct yuv8_video_struct* video_in, struct yuv8_video_struct* video_out);
int yuv8_444to420(struct yuv8_video_struct* video_in, struct yuv8_video_struct* video_out);
int yuv8_444to422(struct yuv8_video_struct* video_in, struct yuv8_video_struct* video_out);
```

### Binary Image/Video Files

The `video_utils.h` file declares functions that help load and save generalized video files in raw, uncompressed format. These functions effectively serialize the video_struct structure:

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

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.

These functions are used to dynamically allocate and free memory for video structure storage:

```
int alloc_video_buff(struct video_struct* video );
void free_video_buff(struct video_struct* video );
```

## Compiling on 32-bit and 64-bit Windows Platforms

Precompiled library `v_osd_v3_0_bitacc_cmodel.lib`, top level demonstration code `run_bitacc_cmodel_config.c` and example code `run_bitacc_cmodel.c` must be compiled with an ANSI C compliant compiler under Windows 32-bit or Windows 64-bit. This section describes an example using Microsoft Visual Studio.

In Visual Studio create a new, empty Win32 Console Application project. As existing items, add:

- `libIpv_osd_v3_0_bitacc_cmodel.lib` to the “Resource Files” folder of the project
- `run_bitacc_cmodel.c` or the `run_bitacc_cmodel_config.c` to the “Source Files” folder of the project
- `v_osd_v3_0_bitacc_cmodel.h` header file to the “Header Files” folder of the project
- `bmp_utils.h` file to the “Header Files” folder of the project
- `rgb_utils.h` file to the “Header Files” folder of the project
- `video_fio.h` file to the “Header Files” folder of the project
- `video_utils.h` file to the “Header Files” folder of the project
- `yuv_utils.h` file to the “Header Files” folder of the project

To build the x64 executable for 64-bit Windows platforms, perform these steps. These steps can be skipped if building the Win32 executable.

1. Right-click on the solution in the Solution Explorer and click **Properties** at the bottom of the pop-up menu.

2. Click **Configuration Manager**.
3. In the Active solution platform drop-down box, select **<New…>**.
4. In the new platform drop-down box, select **x64** and click **OK**.

   Make sure that all the projects now have x64 as the default platform in the Configuration Manager.
5. After the project is created and populated, it must be compiled and linked (built) to create a Win32 or x64 executable. To perform the build step, select **Build Solution** from the Build menu. An executable matching the project name is 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.

***Note:*** The `run_bitacc_cmodel.c` file is an example demonstration that reads no input but generates an output .yuv file from internally generated test patterns. The `run_bitacc_cmodel_config.c` file is a configurable demonstration and requires several input files to run. See Running the Executables for information on command line arguments and input file formats.

## Compiling under 32-bit and 64-bit Linux Platforms

### Example Demonstration

To compile the example demonstration, go to the directory where the header files, the library files and `run_bitacc_cmodel.c` were unpacked. The libraries and header files are referenced during the compilation and linking process. In this directory, perform these steps:

1. Set your LD_LIBRARY_PATH environment variable to include the root directory where the model zip file was unzipped. For example:

   ```
   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_osd_v3_0_bitacc_cmodel.so
   ```
3. In the root directory, compile using the GNU C Compiler by typing this command at the shell prompt:

   ```
   gcc -m32 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
   run_bitacc_cmodel -L. -lIp_v_osd_v3_0_bitacc_cmodel -Wl,-rpath,.

   gcc –m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
   run_bitacc_cmodel -L. -lIp_v_osd_v3_0_bitacc_cmodel -Wl,-rpath,.
   ```
4. This results in the creation of the executable run_bitacc_cmodel, which can be run using this command:

   ```
   ./run_bitacc_cmodel
   ```

   A make file is also included that runs GCC. To clean the executable and compile the example code, enter this command at the shell prompt:

   ```
   make clean all
   ```

### Configurable Demonstration

To compile the configurable demonstration, go to the directory where the header files, the library files and `run_bitacc_cmodel_config.c` were unpacked. The libraries and

header files are referenced during the compilation and linking process. In this directory, perform these steps:

1. Set your LD_LIBRARY_PATH environment variable to include the root directory where the model zip-file was unzipped. For example:

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

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

   libstlport.so.5.1

   libIp_v_osd_v3_0_bitacc_cmodel.so

3. In the root directory, compile using the GNU C Compiler by entering this command at the shell prompt:

   ```
   gcc -x c++ run_bitacc_cmodel_config.c -o run_bitacc_cmodel_config -L.
   -lIp_v_osd_v3_0_bitacc_cmodel -Wl,-rpath,.
   ```

4. This results in the creation of the executable run_bitacc_cmodel, which can be run using this command:

   ```
   ./run_bitacc_cmodel_config -i <Config Filename> <-parameter=value ...>
   ```

   A make file is also included that runs GCC. To clean the executable and compile the example code, enter this following command at the shell prompt:

   ```
   make clean run_bitacc_cmodel_config
   ```

## Running the Executables

Included in the zip file are precompiled executable files for use with 32-bit and 64-bit Windows and Linux platforms. The instructions for running on each platform are included in this section.

### Example Demonstration

The example demonstration does not use command line parameters. To run on a 32-bit or 64-bit Linux platform, perform these steps:

1. Set your $LD_LIBRARY_PATH environment variable to include the root directory where the model zip file was unzipped. For example:

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

2. Copy these files from the /lin64 (for 64-bit Linux) or from the /lin (for 32-bit Linux) directory to the root directory:

   libstlport.so.5.1

   libIp_v_osd_v3_0_bitacc_cmodel.so

   run_bitacc_cmodel

3. Execute the model. From the root directory, enter this command at a shell prompt:

   ```
   run_bitacc_cmodel
   ```

To run on a 32-bit or 64-bit Windows platform, perform these steps:

1. Copy this file from the /nt64 (for 64-bit Windows) or from the /nt (for 32-bit Windows) directory to the root directory:

   ```
   run_bitacc_cmodel.exe
   ```

2. Execute the model. From the root directory, enter this command at a DOS prompt:

   ```
   run_bitacc_cmodel
   ```

During successful execution, the `test.yuv` file is created in the directory containing the run_bitacc_cmodel executable. This file is a planar YUV file in 4:4:4 format. The example demonstration is set up to generate three frames of video data at 1280x720 resolution. Each frame contains the output of three video layers and background color.

Figure E-1 shows frame 1 of the `test.yuv` file. The image shows a background color of orange, a video layer with a horizontal ramp, another video layer with random data, and a graphics controller layer with text and boxes.

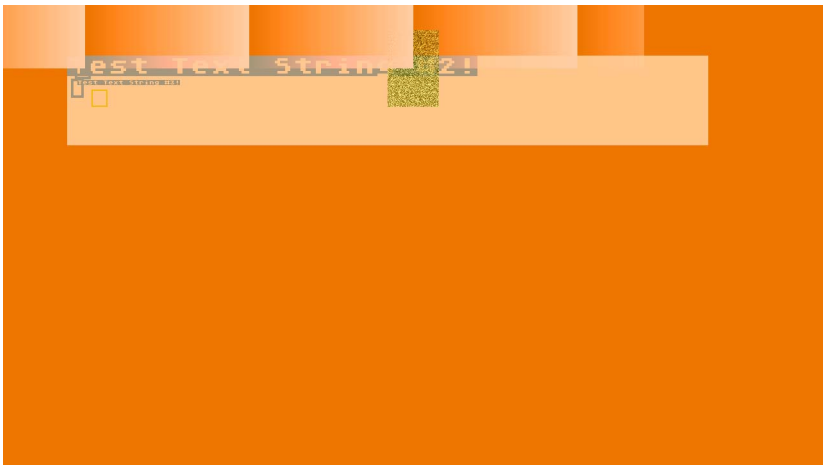


*Figure E-1:* **Example Demonstration Output Image**

## Configurable Demonstration

The configurable demonstration takes multiple command line parameters. To run on a 32-bit or 64-bit Linux platform, perform these steps:

1. Set your $LD_LIBRARY_PATH environment variable to include the root directory where the model zip-file was unzipped. For example:

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

2. Copy these files from the /lin64 (for 64-bit Linux) or from the /lin (for 32-bit Linux) directory to the root directory:

   libstlport.so.5.1

   libIp_v_osd_v3_0_bitacc_cmodel.so

   run_bitacc_cmodel_config

3. Execute the model. From the root directory, enter this command at a shell prompt:

   ```
   run_bitacc_cmodel_config -i <Config Filename> <-parameter=value …>
   ```

To run on a 32-bit or 64-bit Windows platform, perform these steps:

1. Copy this file from the /nt64 (for 64-bit Windows) or from the /nt (for 32-bit Windows) directory to the root directory:

   run_bitacc_cmodel_config.exe

2. Execute the model. From the root directory, enter this command at a DOS prompt:

   ```
   run_bitacc_cmodel_config -i <Config Filename> <-parameter=value …>
   ```

The configurable demonstration reads parameters from the config file specified with the -i <config_file> argument where <config_file> is the relative path and filename of the config file. See Config File Format for more information. Parameters in the config file can be overridden on the command line by prefixing the parameter with a dash ('-') and removing white spaces. For example, the number of frames to simulate can be overridden with this command line argument "-T_NUM_FRAMES=2". Config parameters set on the command line must be set after the -i argument to take effect.

Figure E-2 shows frame 1 of the output of the configurable demonstration from this command line:

```
run_bitacc_cmodel_config -i examples/example0.cfg
```

The image shows a background color of green, a video layer with a horizontal ramp and another video layer with random data.

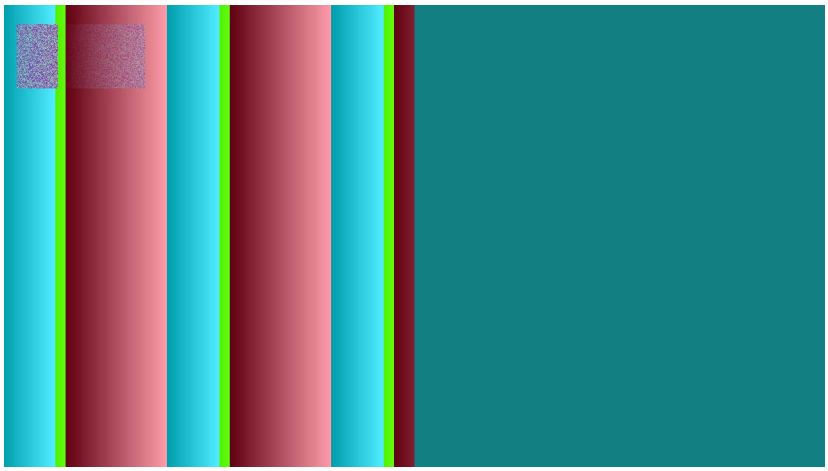


*Figure E-2:* **Configurable Demonstration Output Image (Example 0)**

Figure E-3 shows frame 1 of the output of the configurable demonstration from this command line:

```
run_bitacc_cmodel_config -i examples/example1.cfg
```

The image shows a background color of grey, a video layer with a horizontal ramp and another video layer with a vertical ramp. Each ramp layer (vertical and horizontal) have different ramp starting values for each color component.

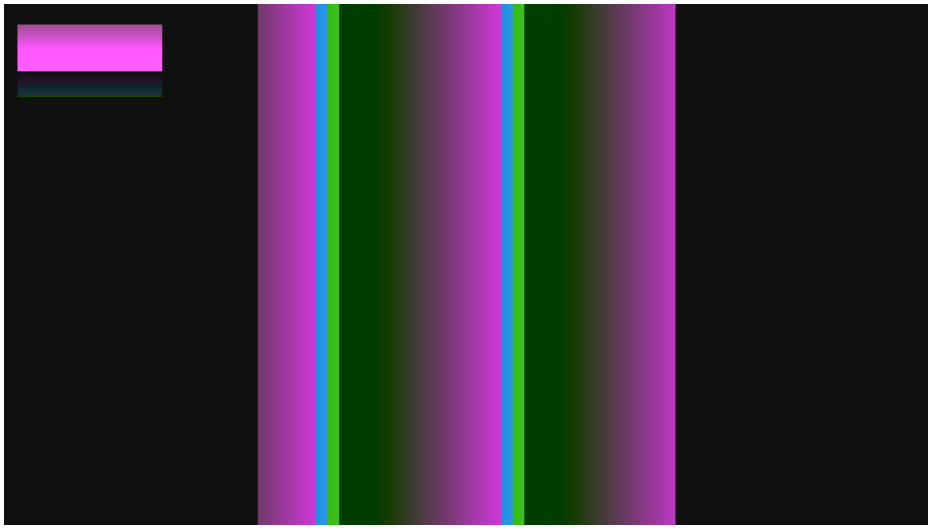


*Figure E-3:* **Configurable Demonstration Output Image (Example 1)**

Figure E-4 shows frame 1 of the output of the configurable demonstration from this command line:

```
run_bitacc_cmodel_config -i examples/example2.cfg
```

The image shows a background color of red, a video layer from a BMP file input and another video layer with random data.

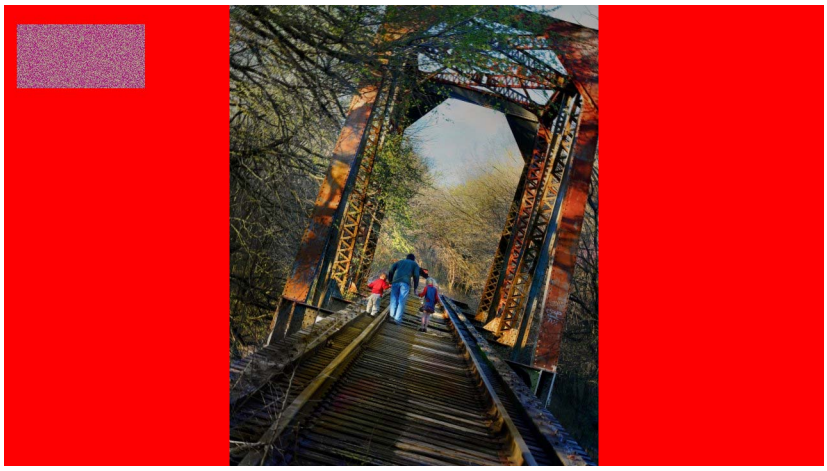


*Figure E-4:* **Configurable Demonstration Output Image (Example 2)**

Figure E-5 shows frame 1 of the output of the configurable demonstration from this command line:

```
run_bitacc_cmodel_config -i examples/example3.cfg
```

The image shows a background color of grey, a video layer from a BMP file input, six other video layers with checkerboard, horizontal ramp and vertical ramp patterns. One graphics controller layer is also displayed generating multi-colored lines, boxes and text.

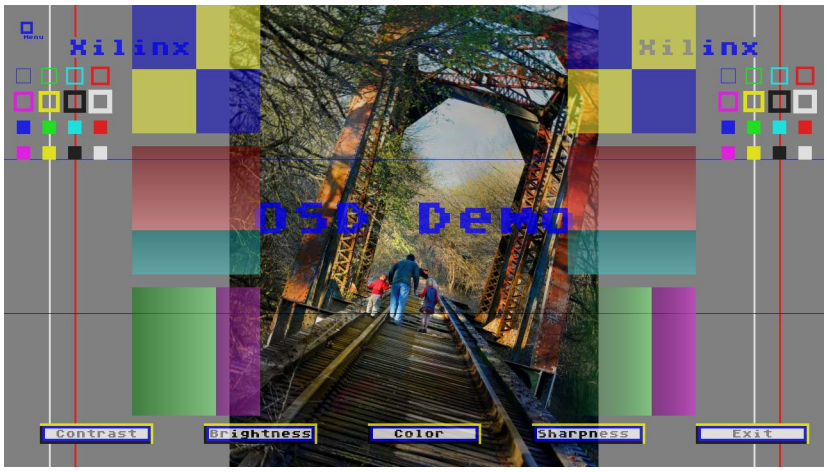



*Figure E-5:* **Configurable Demonstration Output Image (Example 3)**

XILINX®

*Appendix F*

# *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.

## References

These documents provide supplemental material useful with this user guide:

- *AMBA AXI4-Stream Protocol Specification*
- PCI-SIG Documentation (www.pcisig.com/specifications)
  - *PCI Express Base Specification 1.1*
  - *PCI Express Card Electromechanical (CEM) Specification 1.1*

## Technical Support

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

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

- New Features
- Resolved Issues
- Known Issues

## Ordering Information

The LogiCORE IP Video On-Screen Display 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.

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