LogiCORE™ IP On-Screen Display v1.0

User Guide

UG684 September 16, 2009

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/16/09	1.0	Initial Xilinx release.

Table of Contents

Revision History . . . 3

Preface: About This Guide

Guide Contents . . . 11
- Additional Resources . . . 11

Conventions . . . 12
- Typographical . . . 12
- Online Document . . . 13

Chapter 1: Introduction

About the Core . . . 15

Recommended Experience . . . 15

Additional Core Resources . . . 15
- Documentation . . . 15
- Technical Support . . . 15

Providing Feedback . . . 16
- Core . . . 16
- Documentation . . . 16

Nomenclature . . . 16

Chapter 2: Overview

Application . . . 17
- Typical Uses . . . 17

Chapter 3: Algorithm

Alpha-Compositing and Alpha-Blending . . . 19

Chapter 4: Implementation

Basic Architecture . . . 21
- Alpha-Blending Pipeline . . . 23
- Graphics Controller . . . 25

Chapter 5: Use Models

Multi-FIFO to Stream Mode . . . 27

Multi-FIFO to FIFO Mode . . . 28

Chapter 6: IO Interface and Timing

Read FIFO Interface . . . 29

Write FIFO Interface . . . 29

Xilinx Streaming Video Interface (XSVI) . . . 30

Graphics Controller Host Interface 31
Interrupts 31
General Purpose Processor Interface 32
EDK pCore (PLB) Interface 33
Parameter Modification in CORE Generator 33

Chapter 7: Programming the Graphics Controller(s)

Instruction RAM 35
Draw Box (Opcode 1010) 38
Draw Text (Opcode 1110) 38
Draw Box-Text (Opcode 1111) 39
NOP (Opcode 1000) 39
END (Opcode 0000) 39
Color RAM 42
Font RAM 44
Text RAM 47

Chapter 8: EDK pCore Programmers Guide

pCore Address Map 49
pCore Device Driver 58

List of Figures

Preface: About This Guide

Chapter 1: Introduction

Chapter 2: Overview

Figure 2-1: **Example OSD Output** 18

Chapter 3: Algorithm

Chapter 4: Implementation

Figure 4-1: **OSD Block Diagram** 22
Figure 4-2: **Alpha-Blending Pipeline Flow Chart** 24
Figure 4-3: **OSD Graphics Controller Block Diagram** 25

Chapter 5: Use Models

Figure 5-1: **Multi-FIFO to Stream Mode** 27
Figure 5-2: **Multi-FIFO to FIFO Mode** 28

Chapter 6: IO Interface and Timing

Figure 6-1: **Read FIFO Interface Timing** 29
Figure 6-2: **Write FIFO Interface Timing** 30
Figure 6-3: **XSVI Interface Timing** 30
Figure 6-4: **Graphics Controller Host Interface Timing** 31
Figure 6-5: **Interrupt Controller Processor Peripherals** 32

Chapter 7: Programming the Graphics Controller(s)

Figure 7-1: **8x8 1-bit per Pixel Font Example** 44
Figure 7-2: **16x16 2-bits per Pixel Font Example** 45

Chapter 8: EDK pCore Programmers Guide

List of Tables

Preface: About This Guide

Chapter 1: Introduction

Chapter 2: Overview

Chapter 3: Algorithm

Chapter 4: Implementation

Chapter 5: Use Models

Chapter 6: IO Interface and Timing

Chapter 7: Programming the Graphics Controller(s)

Table 7-1: **OSD Instruction Format** 35
Table 7-3: **OSD Instruction Word 1** 36
Table 7-2: **OSD Instruction Word 0** 36
Table 7-4: **OSD Instruction Word 2** 37
Table 7-5: **OSD Instruction Word 3** 38
Table 7-6: **Example Graphics Controller Instruction List (2 Boxes and 1 Box-Text)** 39
Table 7-7: **Example OSD Instruction List (2 Boxes 1 TextBox)** 41
Table 7-8: **Example Color RAM Memory Map (16-Colors)** 42
Table 7-9: **Example Color RAM Host Processor Writes** 43
Table 7-10: **Font RAM Memory Map** 45
Table 7-11: **Example Color RAM Host Processor Writes** 46
Table 7-12: **Example Text RAM Memory Map** 47
Table 7-13: **Example Text RAM Host Processor Writes** 48

Chapter 8: EDK pCore Programmers Guide

Table 8-1: **EDK pCore Address Map** 49
Table 8-2: **OSD Control Register (Address Offset 0x0000)** 51
Table 8-4: **OSD Background Color Register (Address Offset 0x0014)** 52
Table 8-5: **OSD Background Color Register (Address Offset 0x0018)** 52
Table 8-6: **OSD Background Color Register (Address Offset 0x001C)** 52
Table 8-3: **OSD Screen Size Register (Address Offset 0x0010)** 52
Table 8-7: **OSD Layer 0 Control Register (Address Offset 0x0020)** 53
Table 8-8: **OSD Layer 0 Position Register (Address Offset 0x0024)** 53

Table 8-9: **OSD Layer 0 Size Register (Address Offset 0x0028)** 53
Table 8-10: **OSD GC Write Bank Address Register (Address Offset 0x00A0)** 54
Table 8-11: **OSD GC Active Bank Address Register (Address Offset 0x00A4)** 55
Table 8-13: **OSD Software Reset Register (Address Offset 0x100)** 56
Table 8-14: **OSD ISR (Interrupt Status/Clear) Register (Address Offset 0x0220)**. 56
Table 8-12: **OSD GC Data Register (Address Offset 0x00A8)** 56
Table 8-15: **OSD IER (Interrupt Enable) Register (Address Offset 0x0228)** 57
Table 8-16: **Device Driver Source Files**. 58

About This Guide

The *LogiCORE™ IP Video On-Screen Display v1.0 User Guide* provides information about generating the Video On-Screen Display core, customizing and simulating the core using the provided example design, and running the design files through implementation using the Xilinx tools.

Guide Contents

This manual contains the following chapters:

- Chapter 1, "Introduction," introduces the Xilinx Video On-Screen Display core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Overview," illustrates examples of Video On-Screen Display applications.
- Chapter 3, "Algorithm," explains basic theory behind the alpha-blending of video concepts.
- Chapter 4, "Implementation," elaborates on the internal structure in the core and describes interfacing. The Xilinx CORE Generator™ Graphical User Interface is also described.
- Chapter 5, "Use Models," illustrates two likely usage scenarios for the Video On-Screen Display.
- Chapter 6, "IO Interface and Timing," describes correct operation of the input signals and the timing of the output signals. Timing diagrams are included.
- Chapter 7, "Programming the Graphics Controller(s)," explains how to program the graphics controller and describes the format of internal memory storage.
- Chapter 8, "EDK pCore Programmers Guide," introduces the concept of software control of the Video On-Screen Display and provides API details.

Additional Resources

To find additional documentation, see the Xilinx website at:

www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

www.xilinx.com/support/mysupport.htm.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File →Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**`lowpwr =`**{**`on`**|**`off`**}
Vertical bar |	Separates items in a list of choices	**`lowpwr =`**{**`on`**|**`off`**}
Angle brackets < >	User-defined variable or in code samples	<directory name>

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Convention	Meaning or Use	Example
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	**usr_teof_n** is active low.

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Introduction

This chapter introduces the Xilinx On-Screen Display core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.

About the Core

The Video On-Screen Display core is a Xilinx CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see the Video On-Screen Display Product Page. For information about licensing options, see the Video On-Screen Display data sheet, DS728, also available from the Video On-Screen Display Product Page.

Recommended Experience

Although the Video On-Screen Display core is a fully verified solution, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, previous experience building high performance, pipelined FPGA designs using Xilinx implementation software and UCF is recommended.

Contact your local Xilinx representative for a closer review and estimation for your specific requirements

Additional Core Resources

For detailed information about Video On-Screen Display technology and updates to the Video On-Screen Display core, see the following:

Documentation

From the Video On-Screen Display Product Page:

- *Video On-Screen Display Data Sheet*
- *Video On-Screen Display Release Notes*

Technical Support

For technical support, visit www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the Video On-Screen Display core.

Xilinx will provide technical support for use of this product as described in the *LogiCORE™ IP Video On-Screen Display User Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Providing Feedback

Xilinx welcomes comments and suggestions about the Video On-Screen Display core and the documentation supplied with the core.

Core

For comments or suggestions about the Video On-Screen Display core, submit a WebCase from www.xilinx.com/support. Be sure to include the following information:

- Product Name
- Core version number
- Explanation of your comments

Documentation

For comments or suggestions about this document, submit a WebCase from www.xilinx.com/support. Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

Nomenclature

The following are defined for the purposes of this document:

Term	Definition
Color Component	One value from the set of values used to represent the given color space. Depending on the encoding, this value could represent Y, Cb, Cr, R, G, B, etc.
Alpha	The opacity or transparency component.
Channel	The input and output data bus slice that contains color component or alpha values. Typically a channel carries data for one color component, but in the case of the YUV 4:2:2 format, the Cb and Cr color components are sent via only one channel.
RGBA	The color space that encodes color with the Red, Green and Blue color components as well as the alpha component for transparency.
YUVA	The color space that encodes color with the Luma and color difference color components as well as the alpha component for transparency.

XILINX®

Chapter 2

Overview

The Xilinx On-Screen Display (OSD) LogiCORE™ IP provides a flexible video processing block for alpha-blending multiple video images with real-time programmable transparency as well as for generating simple transparent text and graphics overlays. Each video image and graphics overlay is assigned to a layer which can be arranged in Z-plane order (controlling which layers are on top or on bottom). Support is provided for up to eight layers using a combination of external video image inputs (from frame buffer or from FIFO) and internal graphics controllers (including text generators). It supports image sizes up to 4096x4096 pixels in YUVA (4:4:4 or 4:2:2) and RGBA image formats up to 60 fps. Output data is formatted for easy integration with displays or other video systems.

The core is programmable through a comprehensive register interface for setting and controlling 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. The LogiCORE IP is provided with two different interfaces: General Purpose Processor and EDK pCore (including a device driver).

Application

Applications range from broadcast, consumer, automotive, medical imaging and high-resolution video effects to video surveillance, video conferencing and image compression.

Typical Uses

The following example provides details of one scenario for the OSD core (see Figure 2-1). Here we show an example OSD output with multiple video and graphics layers. The three video layers (Video 1, 2 and 3) can be a still image 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 (as shown with the yellow text and menu buttons) and are also blended with the other layers. One other 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.

Figure 2-1: **Example OSD Output**

There are any number of potential applications and/or scenarios for using this core. Other scenarios could include but not limited to the following usages:

- Blending multiple live video or still images
- Generating a heads-up display
- Creating a picture-in-picture (PIP) window
- Drawing a set-top box GUI/menu
- Dynamically placing and moving video images

This list is not meant to be all inclusive, but rather provide examples of other applications.

XILINX®

Chapter 3

Algorithm

This section explains the theory behind 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, refer to 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.

There exist different algebraic compositing algorithms which define different image blending operations. These operations range from "over," "in," "out," "atop," to "xor" and other logical operations. For the purpose of this design, we are only concerned with 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, or conceptually, each layer resides closer or farther from the observer having 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 as follows. Notice that 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 is the same as saying 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.

XILINX®

Chapter 4

Implementation

This chapter elaborates on the internal structure in the core and describes interfacing.

Basic Architecture

The Xilinx On-Screen Display LogiCORE™ IP reads 2D video image data in raster order from up to 8 sources. Each data source can be configured to be a FIFO interface or one of the internal graphics controllers within the OSD core. If a FIFO interface is selected, ports on the OSD are available for connecting to and reading data from the Xilinx Video Frame Buffer Controller or from the Video Direct Memory Access Controller. These ports are also generic enough for easy integration with any FIFO. See the "Port Descriptions" table in the Xilinx Video On-Screen Display data sheet, DS728, available from the Video On-Screen Display Product Page, for more information on the FIFO interface ports. 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 the alpha-blending algorithm mentioned in Chapter 3, "Algorithm." 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 a Xilinx Streaming Video Interface (XSVI). If a FIFO interface is used, the output FIFO almost full flag and the input FIFO empty flags (from each FIFO input source) will halt operation of the OSD until cleared. If an XSVI interface 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, the input FIFO empty flags are ignored. Care must be taken to make sure each input FIFO does not underflow.

Only one output interface must be selected. The OSD cannot drive both interfaces at the same time. See the "Port Descriptions" table in the Xilinx Video On-Screen Display data sheet, DS728, available from the Video On-Screen Display Product Page, for more information on both the output FIFO (VFBC Write Data Interface) and the output XSVI interfaces.

The Xilinx On-Screen Display also requires an input Xilinx Streaming Video Interface. The OSD does not receive video data from this interface. This interface is used only 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 output XSVI interface for use with driving external display hardware. The same is true for the active video signal and is only used 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.

See the Xilinx Video Timing Controller data sheet for more information on video timing signals.

The use of specific video timing signals requires that the input data is always present when requested (the OSD will not look at the input FIFO empty flags) if an XSVI interface is used. If an output FIFO interface is used, it is required that the input and output FIFO flags 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.

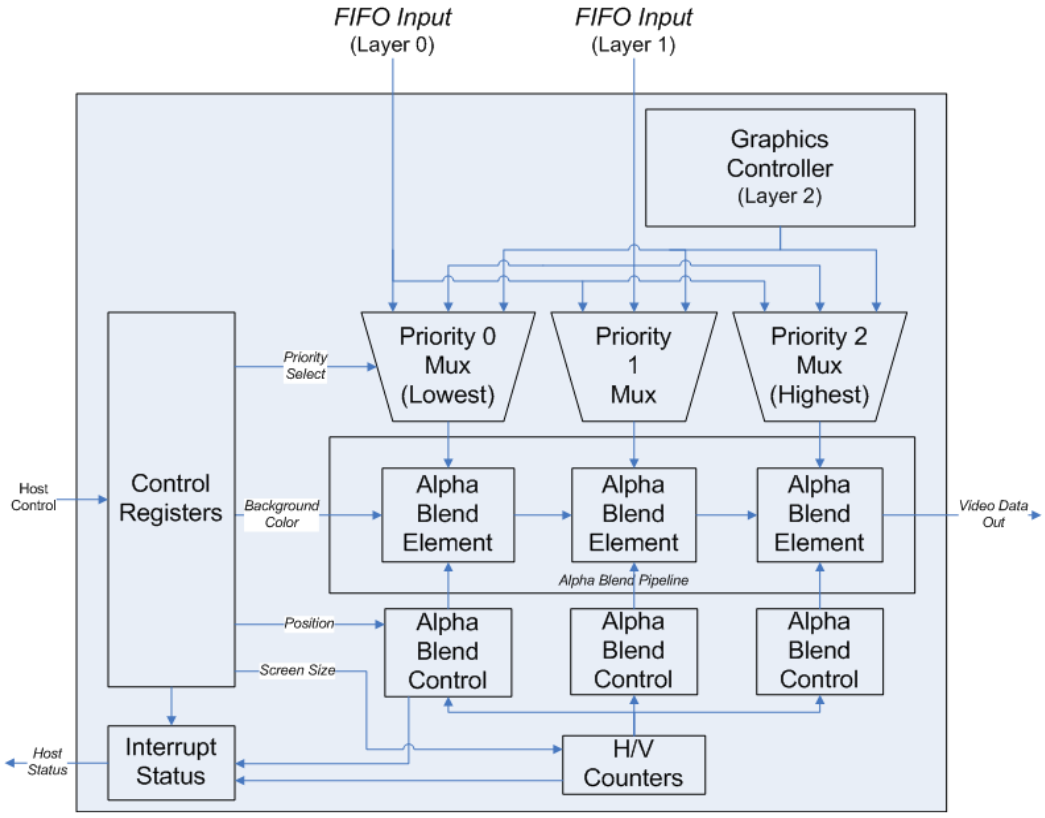

Figure 4-1: **OSD Block Diagram**

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, input FIFO underflow, output FIFO overflow 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, page 24.

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.

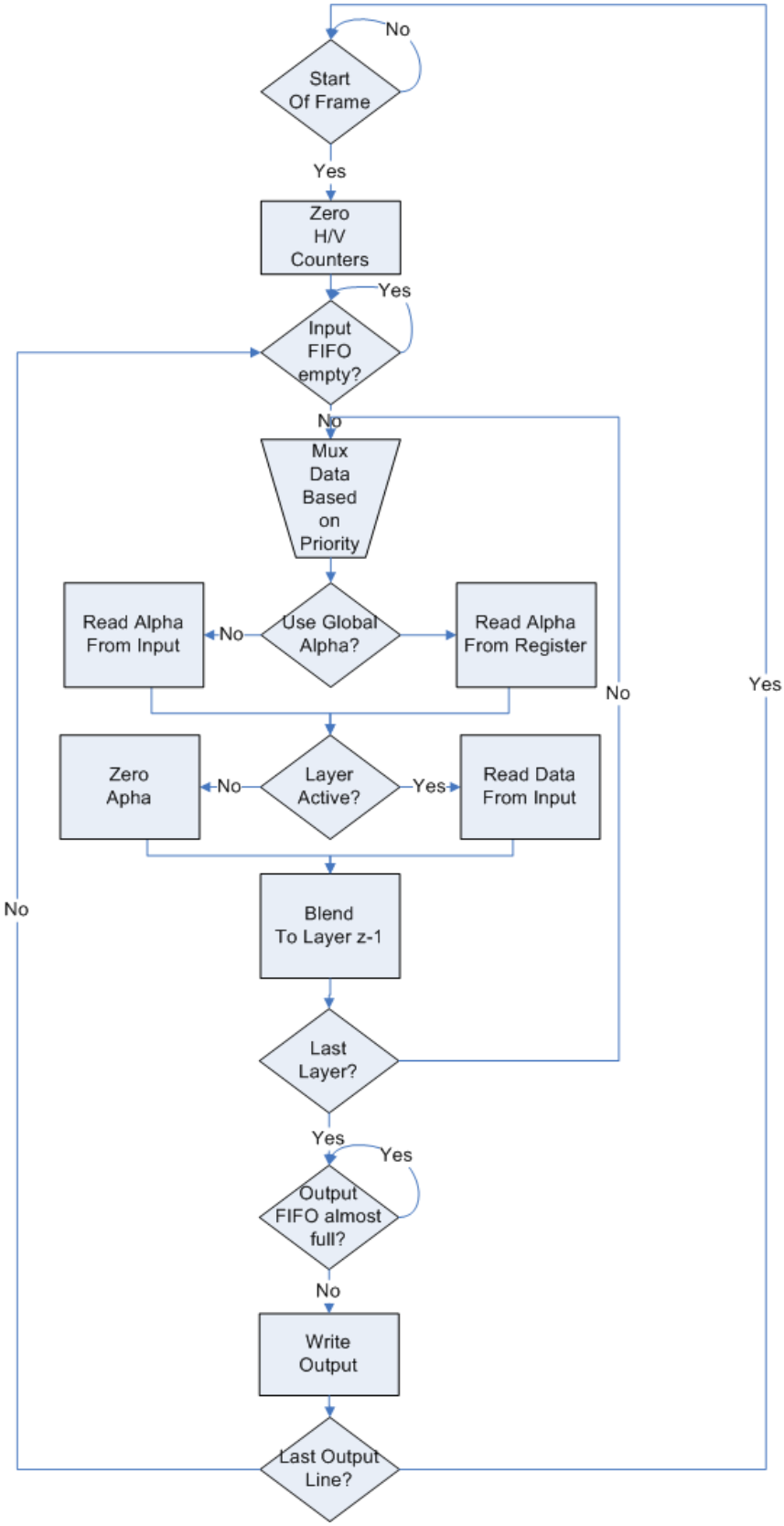

Figure 4-2: **Alpha-Blending Pipeline Flow Chart**

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.

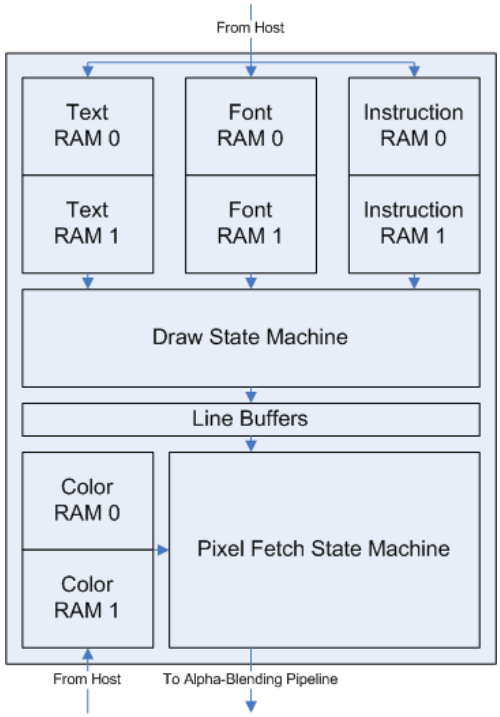

Figure 4-3: **OSD Graphics Controller Block Diagram**

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™ GUI. See "Instruction RAM" in Chapter 7, "Programming the Graphics Controller(s)."

On every 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 as is being written to by the draw state-machine. See Chapter 7, "Programming the Graphics

Controller(s),” for a more detailed description of the graphics controller and storage format of each memory.

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 the “CORE Generator GUI Field Descriptions” table in the Xilinx Video On-Screen Display data sheet, DS728, available from the Video On-Screen Display Product Page, for more information on the graphics controller parameters.

Chapter 5

Use Models

This chapter illustrates two usage scenarios for the Xilinx Video On-Screen Display.

Multi-FIFO to Stream Mode

In this scenario, the Xilinx Video On-Screen Display reads data from multiple input FIFOs and outputs video data in streaming format. 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. See Figure 5-1.

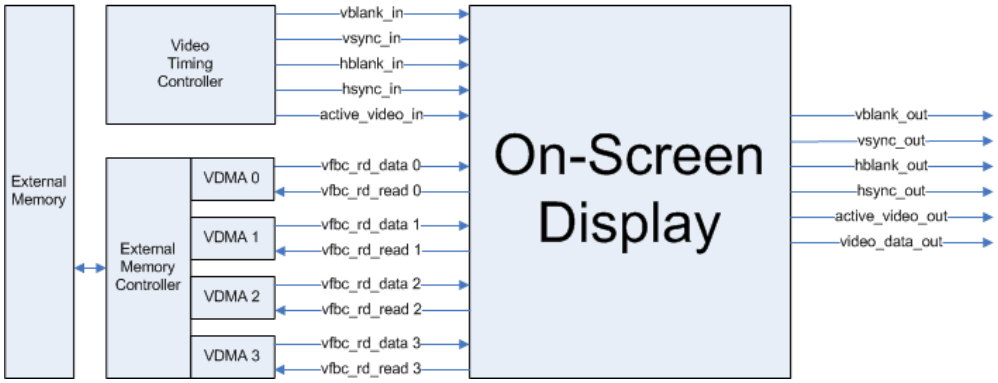

Figure 5-1: **Multi-FIFO to Stream Mode**

Notice that 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:

- The Video Direct Memory Access (VDMA) Data Sheet DS730, available from the Video DMA Core product page
- The Video Timing Controller Data Sheet DS729, available from the Video Timing Controller product page

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, 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. See Figure 5-2.

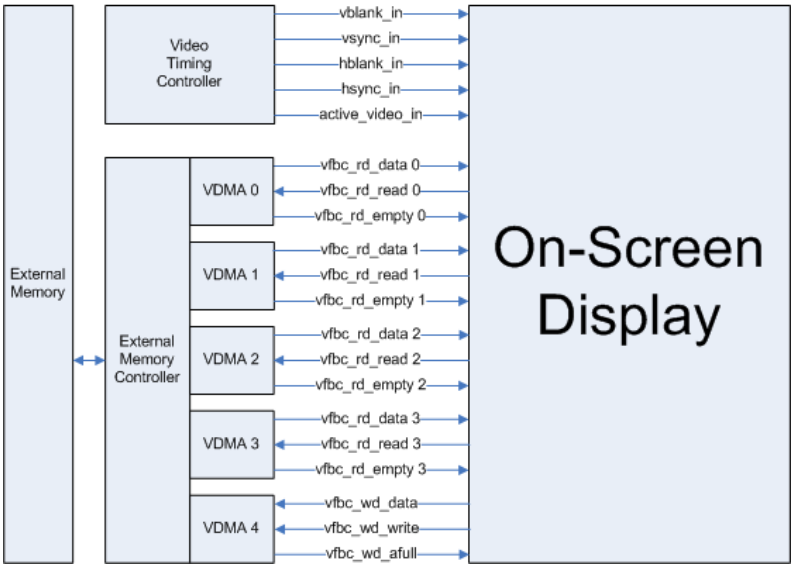

Figure 5-2: **Multi-FIFO to FIFO Mode**

Notice that 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.

Chapter 6

IO Interface and Timing

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

Read FIFO Interface

The Xilinx Video On-Screen Display can be configured to have up to eight Read FIFO Interfaces. These are simple FIFO interfaces with a read enable output and a read latency of one clock cycle, typically used to connect to a VFBC interface within a video system.

Figure 6-1 shows an example read FIFO transaction for when the number of data channels is set to 2 and the output interface is set to VFBC.

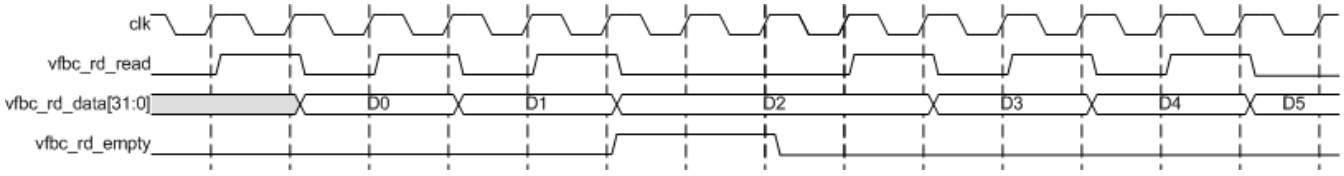

Figure 6-1: **Read FIFO Interface Timing**

The `vfbc_rd_read` signal is the read enable and is driven out by the OSD. The following two or three clock cycles after the read enable is asserted high, the data on the `vfbc_rd_data` bus is read by the OSD. If the OSD is configured for two data channels, the read enable signal will be asserted high every other clock cycle with the data read twice. If the OSD is configured for three data channels, the read enable signal will be asserted high once every three clock cycles with the data read three times. Thus, the data must be held until the next read enable high assertion.

The `vfbc_rd_empty` (empty flag) input is read and the read enable output not asserted only if the output interface of the OSD is configured for a VFBC (Write FIFO) interface.

Write FIFO Interface

The output interface of the Xilinx Video On-Screen Display can be configured to be a VFBC (Write FIFO) interface. In this mode, the OSD has a simple FIFO interface with a write enable output. Valid data is presented on the `vfbc_wd_data` output bus the same clock cycle the `vfbc_wd_write` output is asserted high. See the "Port Descriptions" table in the Xilinx Video On-Screen Display data sheet, DS728, available from the Video On-Screen Display Product Page, for more information on the data format of the `vfbc_wd_data` bus.

Figure 6-2 shows an example write FIFO transaction for when the number of data channels is set to 2.

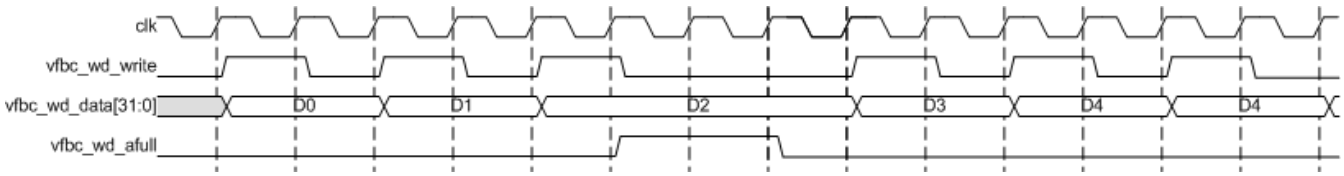

Figure 6-2: **Write FIFO Interface Timing**

Notice that the `vfbc_wd_write` output is asserted high once every other cycle in Figure 6-2. If the number of channels is set to three, then the `vfbc_wd_write` output will be asserted high once every third clock cycle.

The `vfbc_wd_afull` (FIFO almost full flag) input will be monitored and output data will not be written when it is asserted high.

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 five video timing signal outputs. These video timing signals outputs are `vblank_out` (denoting the vertical blanking period), `vsync_out` (denoting the vertical sync), `hblank_out` (denoting the horizontal blanking period), `hsync_out` (denoting the horizontal sync) and `active_video_out` (denoting those clock cycles that contain valid data on the `video_data_out` bus).

Figure 6-3 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.

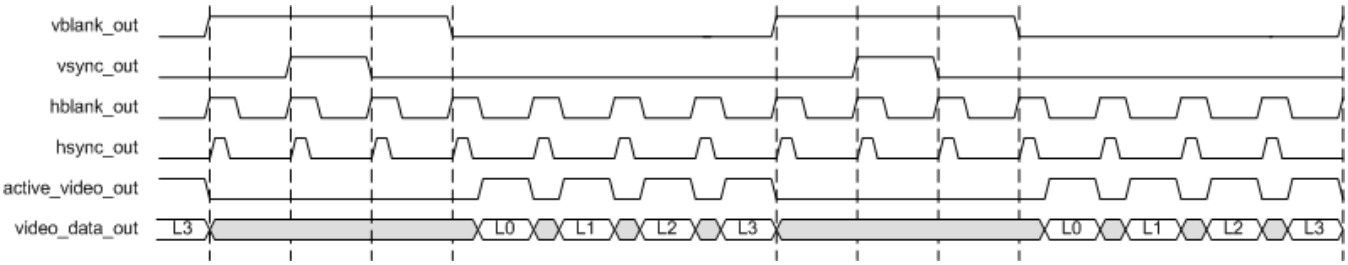

Figure 6-3: **XSVI Interface Timing**

The video timing signal outputs are not generated internally by the OSD. Instead there are five similar video timing signal inputs. These inputs are delayed and present on the XSVI output interface. 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 interface 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. See Table 8-2, "OSD Control Register (Address Offset 0x0000)."

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 in the same 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 same 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 6-4 shows an example graphics controller memory load operation for two memories.

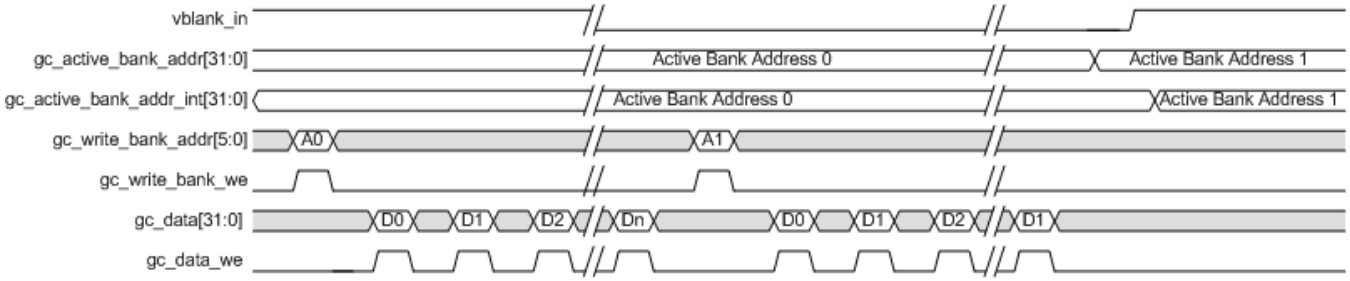

Figure 6-4: **Graphics Controller Host Interface Timing**

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 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.

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 3-31 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 0-2 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 6-5.

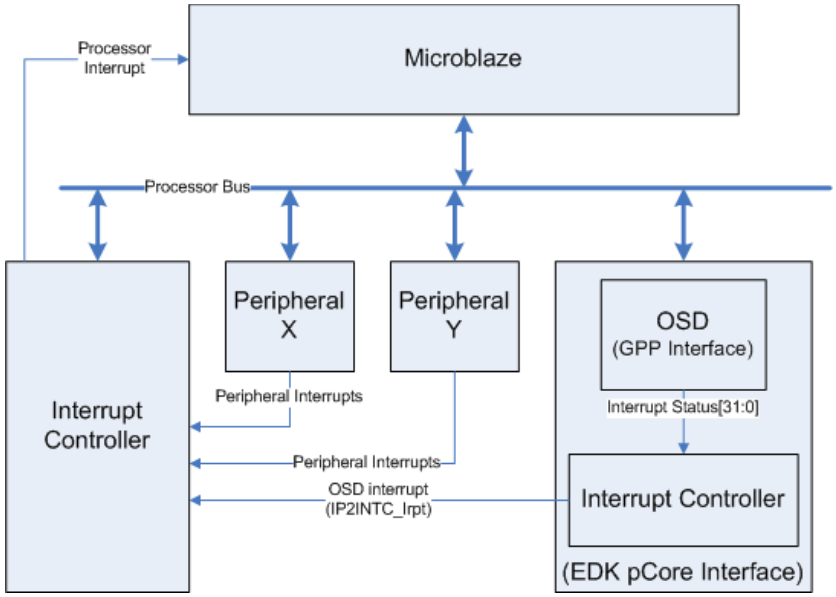

Figure 6-5: **Interrupt Controller Processor Peripherals**

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 the Interrupt Control LogiCORE system.

Interrupt status bits 0–2 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.

See Table 8-2, "OSD Control Register (Address Offset 0x0000)," for more information.

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 a Host Processor via a Register File with 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 the "Port Descriptions" table in the Xilinx Video On-Screen Display data sheet, DS728, available from the Video On-Screen Display Product Page.

EDK pCore (PLB) Interface

The Xilinx Video On-Screen Display, when configured as an EDK pCore, uses the Processor Local Bus to interface to a microprocessor. See the Processor Local Bus (PLB) v4.6 Data Sheet for more information on the PLB 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_v1_0_u0," then the EDK pCore source files will be located in the following directory:

```
<coregen project directory>/v_osd_v1_0_u0/pcores/osd_v1_00_a
```

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

Parameter Modification in CORE Generator

All parameters may be modified in the Xilinx CORE Generator GUI. Modification of these parameters changes the defaults in the .mpd file: osd_v1_00_a/data/osd_v2_1_0.mpd. This provides the user with an alternative to modifying the parameters in the MHS file in the EDK project. **The parameter changes do not get copied into any MHS file. When migrating EDK projects from one place to another, the MPD files for cores modified in this way must also be provided in addition to EDK project files such as the MHS file.** Xilinx recommends that all parameter changes be made in the EDK environment.

Chapter 7

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.

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™ GUI will configure the maximum number of instruction supported by each graphics controller. Table 7-1 shows the OSD Graphics Controller instruction format.

Table 7-1: **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				X1												X0											
1	Reserved																								Text Index							
2	Object Size								Y1												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 7-2, Table 7-3, Table 7-4 and Table 7-5 show the specific bit fields for each instruction word.

Table 7-2: **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				Y0												X0											

Name	Bits	Description
OpCode	31:28	OSD GC OpCode 0000: END 0001 – 01111: Reserved 1000: NOP 1010: Draw Box 1100: Line 1110: Text 1000 – 1110: Reserved 1111: Draw Box-Text
Reserved	27:24	
X1	23:12	Horizontal end pixel of the object. Starts at pixel 0. Not used with Text (opcodes 1110 or 1111).
X0	11:0	Horizontal start pixel of the object. Starts at pixel 0.

Table 7-3: **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																															

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

Table 7-4: **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								Y0											X0												

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. For Opcode 1010 (Draw Box): 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. For Opcode = 1110 (Draw Text): ObjectSize[7:4] (bits 31:28) = Text scale factor 0: Reserved 1: 1x text size 2: 2x text size 4: 4x text size 8: 8x text size For Opcode = 1111 (Draw Box-Text): ObjectSize[3:0] = Line Size (bits 27:24) ObjectSize[7:4] = Text Size (bits 31:28)
Y1	23:12	Vertical end line of the object. Starts at line 0. Must be set to the same value as Y0 for text instruction (opcode 1110).
Y0	11:0	Vertical start line of the object. Starts at line 0.

Table 7-5: **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 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. Currently supporting 16 or 256 colors. 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).

The following is a list of 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-left-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.

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 left-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 7-6 shows an example graphics controller instruction list.

Table 7-6: **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.

Table 7-6: **Example Graphics Controller Instruction List (2 Boxes and 1 Box-Text)**

Address	Data[31:0]	Description
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). Line size of 4. 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 7-6 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 7-7 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 7-7: **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. 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. 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. 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. Lookup color set in address 3 of the internal color LUT BRAM.
0x00A8	0x8000_0000	No-OP.
0x00A8	X	Don't Care.
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.

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

Address Offset	Data[31:0]	Description
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. 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 GUI.

Table 7-8 shows an example Color RAM and its contents with the "Number of Colors" parameter set to 16.

Table 7-8: **Example Color RAM Memory Map (16-Colors)**

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
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

Table 7-8: **Example Color RAM Memory Map (16-Colors) *(Cont'd)***

Address	Data[31:0]	Description
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 7-6 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 7-9 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 7-9: **Example Color RAM Host Processor Writes**

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.

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 7-1 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.

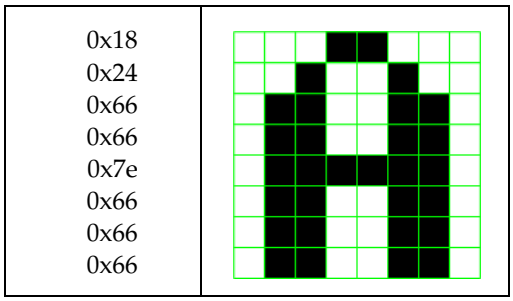

Figure 7-1: **8x8 1-bit per Pixel Font Example**

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 7-8 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 7-2 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.

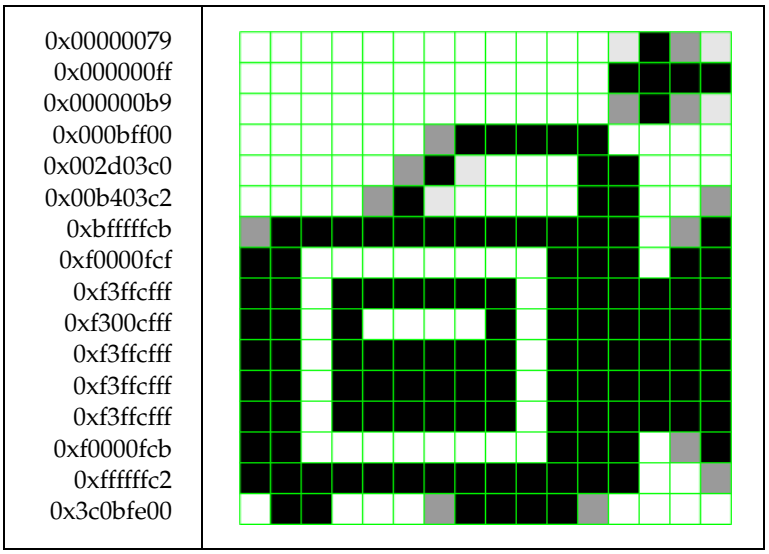

Figure 7-2: **16x16 2-bits per Pixel Font Example**

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 7-8, 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 7-2 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 7-10 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 7-10: **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.

Table 7-10: **Font RAM Memory Map *(Cont'd)***

Address	Data[31:0]	Description
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 7-10 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 7-11 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 7-11: **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.

Table 7-11: **Example Color RAM Host Processor Writes *(Cont'd)***

Address Offset	Data[31:0]	Description
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 128. The maximum string length for each string is configured by the "Maximum String Length" parameter and can be set to any value between 4 and 128. See the "CORE Generator Graphics User Interface (GUI)" section in the Xilinx Video On-Screen Display data sheet, DS728, available from the Video On-Screen Display Product Page, for more information.

Table 7-12 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 7-12: **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"

Table 7-12: **Example Text RAM Memory Map *(Cont'd)***

Address	Data[31:0]	Description
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.

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 7-12 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). his will cause the OSD to use the new text data during the next video frame.

Table 7-13 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 7-13: **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.

XILINX®

Chapter 8

EDK pCore Programmers Guide

This chapter 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 Address Map

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

Table 8-1: **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
0x0044	OSD Layer 2 Position	R/W	Video Layer Position
0x0048	OSD Layer 2 Size	R/W	Video Layer Size
0x004c	Reserved	–	

Table 8-1: **EDK pCore Address Map *(Cont'd)***

Address Offset	Name	Read/ Write	Description
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. Used for all Instantiated Graphics Controllers
0x00a4	OSD GC Active Bank Address	R/W	Graphics Controller Active Bank Addresses. Selected after next vblank. Used for all Instantiated Graphics Controllers
0x00a8	OSD GC Data	R/W	Graphics Controller Data Register Used to write instructions, Character Map, ASCII text strings and color. Used for all Instantiated Graphics Controllers.
0x00aC	Reserved	–	
0x0100	OSD Software Reset	R/W	

Table 8-1: **EDK pCore Address Map *(Cont'd)***

Address Offset	Name	Read/ Write	Description
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 the EDK pCore are big-endian. The registers of the General Purpose Processor Interface are little-endian.

Table 8-2: **OSD Control Register (Address Offset 0x0000)**

0x0000_0000	OSD Control	R/W
Name	**Bits**	**Description**
Reserved	6:31	Reserved
V Blank Polarity	5	INPUT Vertical Blank Polarity 1: The polarity of the v blank input is active high. 0: The polarity of the v blank input is active low.
H Blank Polarity	4	INPUT Horizontal Blank Polarity 1: The polarity of the h blank input is active high. 0: The polarity of the h blank input is active low.
Reserved	3	Reserved
OSD Register Update Enable	2	OSD Register Update Enable Setting this bit to 1 will cause the OSD to re-read all register values after the next frame sync or vertical blank. Setting this bit to 0 will cause the OSD to use its internally buffered register values. 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 This will cause the OSD to start reading from external memory and writing output after the next Vsync.

Table 8-3: **OSD Screen Size Register (Address Offset 0x0010)**

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

Table 8-4: **OSD Background Color Register (Address Offset 0x0014)**

0x0014		R/W
Name	**Bits**	**Description**
Reserved	8:31	
Y/G	0:7	Background Color component of channel 0 Typically, Y (luma) or Green

Table 8-5: **OSD Background Color Register (Address Offset 0x0018)**

0x0018	OSD Background Color Channel 1	R/W
Name	**Bits**	**Description**
Reserved	8:31	
Background Color 1	0:7	Background Color component of channel 1 Typically, U (Cb) or Blue

Table 8-6: **OSD Background Color Register (Address Offset 0x001C)**

0x001c	OSD Background Color Channel 2	R/W
Name	**Bits**	**Description**
Reserved	8:31	
Background Color 2	0:7	Background Color component of channel 2 Typically, V (Cr) or Red

Table 8-7: **OSD Layer 0 Control Register (Address Offset 0x0020)**

0x0020	OSD Layer 0 Control	R/W
Name	**Bits**	**Description**
Reserved	24:31	
Alpha	16:23	Layer 0 Global Alpha Value 0 = Layer 100% transparent 255 = Layer 0% transparent (100% opaque)
Reserved	11:15	
Priority	8:10	Layer 0 Priority 0 = lowest 1 = higher .. 7 = highest
Reserved	2:7	
Layer0_Galpha_en	1	Layer 0 Global Alpha Enable
Layer0_en	0	Layer 0 Enable

Table 8-8: **OSD Layer 0 Position Register (Address Offset 0x0024)**

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

Table 8-9: **OSD Layer 0 Size Register (Address Offset 0x0028)**

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

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

Table 8-10: **OSD GC Write Bank Address Register (Address Offset 0x00A0)**

0x00A0	OSD GC Write Bank Address	R/W
Name	**Bits**	**Description**
Reserved	11:31	
GC Number	8:10	Graphics Controller Number 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	3:7	
GC_Write_Bank_ Addr	0:2	**OSD GC Bank Write Address** Controls which memory bank to write data. 000: Write data into Instruction RAM 0 001: Write data into Instruction RAM 1 010: Write data into Color RAM 0 011: Write data into Color RAM 1 100: Write data into Text RAM 0 101: Write data into Text RAM 1 110: Write data into Font RAM 0 111: Write data into Font RAM 1

Table 8-11: **OSD GC Active Bank Address Register (Address Offset 0x00A4)**

0x00A4	OSD GC Active Bank Address	R/W
Name	**Bits**	**Description**
GC_Char_ActBank	24:31	Sets the Active CharacterMap/Font Bank. Bit 31 = Active Font RAM Bank for GC 7 Bit 30 = Active Font RAM Bank for GC 6 Bit 29 = Active Font RAM Bank for GC 5 Bit 28 = Active Font RAM Bank for GC 4 Bit 27 = Active Font RAM Bank for GC 3 Bit 26 = Active Font RAM Bank for GC 2 Bit 25 = Active Font RAM Bank for GC 1 Bit 24 = Active Font RAM Bank for GC 0
GC_Text_ActBank	16:23	Sets the active Text Bank. Bit 23 = Active Text RAM Bank for GC 7 Bit 22 = Active Text RAM Bank for GC 6 Bit 21 = Active Text RAM Bank for GC 5 Bit 20 = Active Text RAM Bank for GC 4 Bit 19 = Active Text RAM Bank for GC 3 Bit 18 = Active Text RAM Bank for GC 2 Bit 17 = Active Text RAM Bank for GC 1 Bit 16 = Active Text RAM Bank for GC 0
GC_Col_ActBank	8:15	Sets the active Color Table Bank. Bit 15 = Active Color RAM Bank for GC 7 Bit 14 = Active Color RAM Bank for GC 6 Bit 13 = Active Color RAM Bank for GC 5 Bit 12 = Active Color RAM Bank for GC 4 Bit 11 = Active Color RAM Bank for GC 3 Bit 10 = Active Color RAM Bank for GC 2 Bit 09 = Active Color RAM Bank for GC 1 Bit 08 = Active Color RAM Bank for GC 0
GC_Ins_ActBank	0:70	Sets the active Instruction Bank. Bit 07 = Active Instruction RAM Bank for GC 7 Bit 06 = Active Instruction RAM Bank for GC 6 Bit 05 = Active Instruction RAM Bank for GC 5 Bit 04 = Active Instruction RAM Bank for GC 4 Bit 03 = Active Instruction RAM Bank for GC 3 Bit 02 = Active Instruction RAM Bank for GC 2 Bit 01 = Active Instruction RAM Bank for GC 1 Bit 00 = Active Instruction RAM Bank for GC 0

Table 8-12: **OSD GC Data Register (Address Offset 0x00A8)**

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

Table 8-13: **OSD Software Reset Register (Address Offset 0x100)**

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

Table 8-14: **OSD ISR (Interrupt Status/Clear) Register (Address Offset 0x0220)**

0x0220	ISR – Interrupt Status/Clear	R/W
Name	**Bits**	**Description**
OSD GC Address Overflow	24:31	OSD GC Instruction Overflow Interrupt 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). 31: Graphics Controller 7 … 24: Graphics Controller 0
OSD GC_Ins_error	16:23	OSD GC Instruction Error Interrupt 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. 23: Graphics Controller 7 … 16: Graphics Controller 0
Reserved	13:15	
OSD Output Overflow Error	12	OSD Output Overflow Error Interrupt This interrupt is asserted if the output is set to VFBC and the VFBC write enable occurs while the VFBC write FIFO is almost full.
OSD Input Underflow Error	4:11	OSD Input Underflow Error Interrupt This interrupt is asserted if the input is set to VFBC and the VFBC read enable occurs while the VFBC read FIFO is empty. 11: input 7 .. 4: input 0

Table 8-14: **OSD ISR (Interrupt Status/Clear) Register (Address Offset 0x0220)**

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 8-15: **OSD IER (Interrupt Enable) Register (Address Offset 0x0228)**

0x0228	IER – Interrupt Enable	R/W
Name	**Bits**	**Description**
OSD GC Address Overflow	24:31	OSD GC Instruction Overflow Interrupt 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). 31: Graphics Controller 7 … 24: Graphics Controller 0
OSD GC_Ins_error	16:23	OSD GC Instruction Error Interrupt 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. 23: Graphics Controller 7 … 16: Graphics Controller 0
Reserved	13:15	
OSD Output Overflow Error	12	OSD Output Overflow Error Interrupt This interrupt is asserted if the output is set to VFBC and the VFBC write enable occurs while the VFBC write FIFO is almost full.
OSD Input Underflow Error	4:11	OSD Input Underflow Error Interrupt This interrupt is asserted if the input is set to VFBC and the VFBC read enable occurs while the VFBC read FIFO is empty. 11: input 7 .. 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

pCore Device Driver

The Xilinx On-Screen Display pCore includes a software driver written in the C Language that you can use 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 8-16 lists the files that are included with the Xilinx OSD pCore driver and their description.

Table 8-16: **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.