Video Mixer v1.0

LogiCORE IP Product Guide

Vivado Design Suite

PG243 April 5, 2017

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The Xilinx® LogiCORE™ IP Video Mixer core provides a flexible video processing block for alpha blending and compositing multiple video and/or graphics layers. Support for up to eight layers, with an optional logo layer, using a combination of video inputs from either frame buffer or streaming video cores (through AXI4-Stream interfaces) is provided. The core is programmable through a comprehensive register interface to control frame size, background color, layer position, and the AXI4-Lite interface. A comprehensive set of interrupt status bits is provided for processor monitoring.

Features

- Supports (per pixel) alpha-blending of eight video/graphics and logo layers
- Optional logo (in block RAM (BRAM)) layer with color transparency support
- Layers can either be memory mapped AXI4 interface or AXI4-Stream
- Provides programmable background color
- Provides programmable layer position and size
- Provides scaling of layers by 1x, 2x, or 4x
- Optional built-in color space conversion
- Supports RGB, YUV 444, YUV 422, YUV 420
- Supports 8, 10, 12, and 16 bits per color component input and output on stream interface, 8-bit and 10-bit per color component on memory interface
- Supports semi-planar memory formats next to packed memory formats
- Supports spatial resolutions from 64 × 64 up to 4,096 × 2,160
- Supports 4K 60 fps in all supported device families(1)

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families UltraScale™ Architecture Zynq®-7000 All Programmable SoC 7 Series
Supported User Interfaces	AXI4-Master, AXI4-Lite, AXI4-Stream(2)
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Not Provided
Example Design	Yes
Test Bench	Not Provided
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Encrypted RTL
Supported S/W Driver(3)	Standalone DRM/KMS
Tested Design Flows(4)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. Video protocol as defined in the "Video IP: AXI Feature Adoption" section of *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 1].
3. Standalone driver details can be found in the software development kit (SDK) directory (<install_directory>/SDK/<release>/data/embeddedsw/doc/xilinx_drivers.htm). Linux OS and driver support information is available from the Xilinx Wiki page.
4. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

1. Performance on low power devices might be lower.

Overview

The Xilinx® LogiCORE™ IP Video Mixer produces one single output video stream from multiple external video and/or graphics sources. Sources can be dynamically positioned, scaled, and combined using alpha-blending.

Feature Summary

The Video Mixer is a highly configurable IP core that supports blending of up to eight video and/or graphics layers plus an additional logo layer into one single output video stream. All layers except the master layer can either be a memory mapped AXI4 interface or AXI4-Stream based. Alpha-blending (global or per pixel) and scaling is supported per layer. Finally, built-in color space conversion between RGB and YUV 4:4:4 and chroma re-sampling between YUV 4:4:4, YUV 4:2:2, and YUV 4:2:0 is optionally available.

The mixer provides an optional logo layer. It blends a logo that is stored in block RAM on the top-most layer. A programmable color key can be used to make part of the logo transparent. Also, alpha-blending (global or per pixel) can be used for logo transparency.

Alpha-blending is the convex combination of two image layers allowing for transparency. Each layer in the mixer has a fixed Z-plane order; or conceptually, each layer resides closer to 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. This can either be done through a global alpha, that is, every pixel uses the same alpha value, or through a per pixel alpha value in case either the RGBA8, BGRA8, or YUVA8 video format is selected. Note that per-pixel alpha is not supported on streaming layers.

Scaling by means of pixel and line repeat is supported, and provides scaling of layers by 1x, 2x, or 4x. This feature might be used to save on memory bandwidth used by a layer, that is, a memory layer can read in a 1,920 × 1,080 frame buffer while scaling it up on-the-fly to a 3,840 × 2,160 resolution.

The Video Mixer supports parallel processing of multiple pixels per clock cycle allowing support for resolutions beyond 1080p60 with up to three color components, each of 8, 10, 12, or 16 bits.

Applications

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

- Video surveillance
- Machine vision
- Video conferencing
- Set-top box displays

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information on pricing and availability.

For more information, visit the Video Mixer product web page.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

The Video Mixer is compliant with the AXI4-Stream Video Protocol, AXI4-Lite interconnect and memory mapped AXI4 interface standards. For additional information, see the "Video IP: AXI Feature Adoption" section of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 1].

Performance

The following sections detail the performance characteristics of the Video Mixer.

Maximum Frequencies

The following are typical clock frequencies for the target devices:

- Virtex®-7 and Virtex UltraScale™ devices with –2 speed grade or higher: 300 MHz
- Kintex®-7 and Kintex UltraScale™ devices with –2 speed grade or higher: 300 MHz
- Artix®-7 devices with –2 speed grade or higher: 150 MHz

The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the device, using a different version of Xilinx® tools, and other factors.

Throughput

The Video Mixer supports bi-directional data throttling between its AXI4-Stream slave and master interfaces. If the slave side data source is not providing valid data samples (`s_axis_video_tvalid` is not asserted), the core cannot produce valid output samples after its internal buffers are depleted. Similarly, if the master side interface is not ready to accept valid data samples (`m_axis_video_tready` is not asserted) the core cannot accept valid input samples after its buffers become full.

If the master interface is able to provide valid samples (`s_axis_video_tvalid` is High) and the slave interface is ready to accept valid samples (`m_axis_video_tready` is High), typically the core can process and produce one, two, or four pixels specified by **Samples Per Clock** in the Vivado Integrated Design Environment (IDE) per `ap_clk` cycle.

However, at the end of each scan line and frame the core flushes internal pipelines for several clock cycles, during which the `s_axis_video_tready` is deasserted signaling that the core is not ready to process samples.

When the Video Mixer is processing timed streaming video (which is typical for most video sources), the flushing periods coincide with the blanking periods and therefore do not reduce the throughput of the system.

When operating on a streaming video source (that is, not frame buffered data), the Video Mixer must operate minimally at the burst data rate. For example, 148.5 MHz for a 1080p60 video source for a one sample per clock configuration of the IP. For a 4K 60 fps video source, the core must operate at 297 MHz for a two sample per clock configuration, or 148.5 MHz for a four sample per clock configuration on slower devices such as Artix®-7.

Resource Utilization

For full details about performance and resource utilization, visit the Performance and Resource Utilization web page.

Port Descriptions

The Video Mixer uses industry standard control and data interfaces to connect to other system components. The following sections describe the various interfaces available with the core. Figure 2-1 illustrates a Video Mixer I/O diagram. In this configuration, the IP has 10 AXI interfaces:

- AXI4-Lite control interface (`s_axi_CTRL`)
- AXI4-Stream streaming video output (`m_axis_video`)
- AXI4-Stream streaming video input (`s_axis_video`, `s_axis_video1`, etc.)
- Memory mapped AXI4 interface (`m_axi_mm_video4`, `m_axi_mm_video5`, etc.).

Figure 2-1: **Video Mixer I/O Diagram**

The interfaces change depending on the number of layers and layer type, that is, streaming or memory based. For example, Figure 2-2 shows a minimum configuration of the Video Mixer IP with only one layer and a logo layer.

Figure 2-2: **Video Mixer I/O Diagram Single Layer**

Common Interface Signals

Table 2-1 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream, memory mapped AXI4 data, or AXI4-Lite control interfaces.

Table 2-1: **Common Interface Signals**

Signal Name	I/O	Width	Description
ap_clk	I	1	Video core clock
ap_rst_n	I	1	Video core active-Low clock enable
interrupt	O	1	Interrupt Request Pin

The `ap_clk` and `ap_rst_n` signals are shared between the core, the AXI4-Stream, memory mapped AXI4 data interfaces, and the AXI4-Lite control interface.

ap_clk

The AXI4-Stream, memory mapped AXI4, and AXI4-Lite interfaces must be synchronous to the core clock signal `ap_clk`. All AXI4-Stream, memory mapped AXI4 interface input signals and AXI4-Lite control interface input signals are sampled on the rising edge of `ap_clk`. All AXI4-Stream output signal changes occur after the rising edge of `ap_clk`.

ap_rst_n

The `ap_rst_n` pin is an active-Low, synchronous reset input pertaining to both AXI4-Lite, AXI4-Stream, and memory mapped AXI4 interfaces. When `ap_rst_n` is set to 0, the core resets at the next rising edge of `ap_clk`.

interrupt

The interrupt status output bus can be integrated with an external interrupt controller that has independent interrupt enable/mask, interrupt clear, and interrupt status registers that allows for interrupt aggregation to the system processor.

AXI4-Stream Video Interface

The Video Mixer in any configuration has AXI4-Stream video input and output interfaces named `s_axis_video` and `m_axis_video`, respectively. Per additional streaming layer, there are an additional AXI4-Stream video input named `s_axis_videoi` with *i* representing the layer number minus 1. All video streaming interfaces follow the interface specification as defined in the Video IP chapter of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 1]. The video AXI4-Stream interface can be single, dual, or quad pixels per clock and can support 8, 10, 12, or 16 bits per component. The streaming interface configuration (samples per clock and bits per component) is chosen at IP level and applies to all instances of the AXI4-Stream interface.

Table 2-2 through Table 2-4 explain the pixel mapping of an AXI4-Stream interface with 2 pixels per clock and 10 bits per component configuration for all supported color formats. Given that the Video Mixer always requires a hardware configuration for three component video, the AXI4-Stream Subset Converter is needed to communicate with other IPs of two or one component video interface in YUV 4:2:2, 4:2:0, or Luma Only.

Table 2-2: **Dual Pixels Per Clock, 10 Bits Per Component Mapping for RGB**

63:60	59:50	49:40	39:30	29:20	19:10	9:0
Zero padding	R1	B1	G1	R0	B0	G0

Table 2-3: **Dual Pixels Per Clock, 10 Bits Per Component Mapping for YUV 4:4:4**

63:60	59:50	49:40	39:30	29:20	19:10	9:0
Zero padding	V1	U1	Y1	V0	U0	Y0

Table 2-4: **Dual Pixels Per Clock, 10 Bits Per Component Mapping for YUV 4:2:2**

63:60	59:50	49:40	39:30	29:20	19:10	9:0
Zero padding	Zero padding	Zero padding	V0	Y1	U0	Y0

Table 2-5 shows the interface signals for input and output AXI4-Stream video streaming interfaces.

Table 2-5: **AXI4-Stream Interface Signals**

Signal Name	I/O	Width	Description
s_axis_tdata	I	floor(((3 × bits_per_component × pixels_per_clock) + 7) / 8) × 8	Input Data
s_axis_tready	O	1	Input Ready
s_axis_tvalid	O	1	Input Valid
s_axis_tdest	I	1	Input Data Routing Identifier
s_axis_tkeep	I	(s_axis_video_tdata width) / 8	Input byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream.
s_axis_tlast	I	1	Input End of Line
s_axis_tstrb	I	(s_axis_video_tdata width) / 8	Input byte qualifier that indicates whether the content of the associated byte of TDATA is processed as a data byte or a position byte.
s_axis_tuser	I	1	Input Start of Frame
m_axis_tdata	O	floor(((3 × bits_per_component × pixels_per_clock) + 7) / 8) × 8	Output Data
m_axis_tdest	O	1	Output Data Routing Identifier
m_axis_tid	O	1	Output Data Stream Identifier
m_axis_tkeep	O	(m_axis_video_tdata width) / 8	Output byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream.
m_axis_tlast	O	1	Output End of Line
m_axis_tready	I	1	Output Ready
m_axis_tstrb	O	(m_axis_video_tdata width) / 8	Output byte qualifier that indicates whether the content of the associated byte of TDATA is processed as a data byte or a position byte.
m_axis_tuser	O	1	Output Start of Frame
m_axis_tvalid	O	1	Output Valid

All video streaming interfaces run at the IP core clock speed, `ap_clk`.

Memory Mapped AXI4 Interface

Per memory layer, there is a memory mapped AXI4 interface named `m_axi_mm_videoi` with *i* representing the layer number minus 1. The memory mapped AXI4 interface runs on the `ap_clk` clock domain. The signals follow the specification as defined in the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 1]. Table 2-6 shows the pixel formats in memory supported by the Video Mixer.

Table 2-6: **Pixel Formats**

Video Format	Description	Bits per Component	Bytes per Pixel
RGBX8	packed RGB	8	4 bytes per pixel
YUVX8	packed YUV 4:4:4	8	4 bytes per pixel
YUYV8	packed YUV 4:2:2	8	2 bytes per pixel
RGBA8	packed RGB with alpha	8	4 bytes per pixel
BGRA8	packed BGR with alpha	8	4 bytes per pixel
YUVA8	packed YUV 4:4:4	8	4 bytes per pixel
RGBX10	packed RGB	10	4 bytes per pixel
YUVX10	packed YUV 4:4:4	10	4 bytes per pixel
RGB565	packed RGB	5 bits per R component 6 bits per G component 5 bits per B component	2 bytes per pixel
Y_UV8	semi-planar YUV 4:2:2	8	1 byte per pixel per plane
Y_UV8_420	semi-planar YUV 4:2:0	8	1 byte per pixel per plane
Y_UV8	semi-planar YUV 4:2:2	8	1 byte per pixel per plane
RGB8	packed RGB	8	3 bytes per pixel
Y_UV10	semi-planar YUV 4:2:2	10	4 bytes per 3 pixels per plane
Y_UV10_420	semi-planar YUV 4:2:0	10	4 bytes per 3 pixels per plane
Y8	packed luma only	8	1 byte per pixel
Y10	packed luma only	10	4 bytes per 3 pixels

The following tables explain the expected pixel mappings in memory for each of the mentioned listed formats.

RGBX8

Packed RGB, 8 bits per component. Every RGB pixel in memory is represented with 32 bits, as shown. The images need be stored in memory in raster order, that is, top-left pixel first, bottom-right pixel last. Bits[31:24] do not contain pixel information.

31:24	23:16	15:8	7:0
X	B	G	R

YUVX8

Packed YUV 4:4:4, 8 bits per component. Every YUV 4:4:4 pixel in memory is represented with 32 bits, as shown. Bits[31:24] do not contain pixel information.

31:24	23:16	15:8	7:0
X	V	U	Y

YUYV8

Packed YUV 4:2:2, 8 bits per component. Every two YUY 4:2:2 pixels in memory are represented with 32 bits, as shown.

31:24	23:16	15:8	7:0
V0	Y1	U0	Y0

RGBA8

Packed RGB with alpha, 8 bits per component. Every RGB with alpha pixel is represented with 32 bits, as shown. Bits[31:24] contain alpha information, 0 is fully transparent, 255 is fully opaque.

31:24	23:16	15:8	7:0
A	B	G	R

BGRA8

Packed BGR with alpha, 8 bits per component. Every BGR with alpha pixel is represented with 32 bits, as shown. Bits[31:24] contain alpha information, 0 is fully transparent, 255 is fully opaque.

31:24	23:16	15:8	7:0
A	R	G	B

YUVA8

Packed YUV with alpha, 8 bits per component. Every YUV 4:4:4 with alpha pixel is represented with 32 bits, as shown. Bits[31:24] contain alpha information, 0 is fully transparent, 255 is fully opaque.

31:24	23:16	15:8	7:0
A	V	U	Y

RGBX10

Packed RGB, 10 bits per component. Every RGB pixel is represented with 32 bits, as shown. Bits[31:30] do not contain any pixel information.

31:30	29:20	19:10	9:0
X	B	G	R

YUVX10

Packed YUV 4:4:4, 10 bits per component. Every YUV 4:4:4 pixel is represented with 32 bits, as shown. Bits[31:30] do not contain any pixel information.

31:30	29:20	19:10	9:0
X	V	U	Y

RGB565

Packed RGB with 5 bits per R component, 6 bits per G component, 5 bits per B component. Every RGB pixel is represented with 16 bits, as shown.

15:11	10:5	4:0
B	G	R

Y_UV8

Semi-planar YUV 4:2:2 with 8 bits per component. Y and UV stored in separate planes as shown. The UV plane is assumed to have an offset of stride × height bytes from the Y plane buffer address.

31:24	23:16	15:8	7:0
Y3	Y2	Y1	Y0

31:24	23:16	15:8	7:0
V2	U2	V0	U0

Y_UV8_420

Semi-planar YUV 4:2:0 with 8 bits per component. Y and UV stored in separate planes as shown. The UV plane is assumed to have an offset of stride × height bytes from the Y plane buffer address.

31:24	23:16	15:8	7:0
Y3	Y2	Y1	Y0

31:24	23:16	15:8	7:0
V4	U4	V0	U0

RGB8

Packed RGB, 8 bits per component. Every RGB pixel in memory is represented with 24 bits, as shown. The images need be stored in memory in raster order, that is, top-left pixel first, bottom-right pixel last.

23:16	15:8	7:0
B	G	R

YUV8

Packed YUV 4:4:4, 8 bits per component. Every YUV 4:4:4 pixel in memory is represented with 24 bits, as shown. The images need be stored in memory in raster order, that is, top-left pixel first, bottom-right pixel last.

23:16	15:8	7:0
V	U	Y

Y_UV10

Semi-planar YUV 4:2:2 with 10 bits per component. Every 3 pixels is represented with 32 bits. Bits[31:30] do not contain any pixel information. Y and UV stored in separate planes as shown. The UV plane is assumed to have an offset of stride x height bytes from the Y plane buffer address.

63:62	61:52	51:42	41:32	31:30	29:20	19:10	9:0
X	Y5	Y4	Y3	X	Y2	Y1	Y0

63:62	61:52	51:42	41:32	31:30	29:20	19:10	9:0
X	V4	U4	V2	X	U2	V0	U0

Y_UV10_420

Semi-planar YUV 4:2:0 with 10 bits per component. Every 3 pixels is represented with 32 bits. Bits[31:30] do not contain any pixel information. Y and UV stored in separate planes as shown. The UV plane is assumed to have an offset of stride x height bytes from the Y plane buffer address.

63:62	61:52	51:42	41:32	31:30	29:20	19:10	9:0
X	Y5	Y4	Y3	X	Y2	Y1	Y0

63:62	61:52	51:42	41:32	31:30	29:20	19:10	9:0
X	V8	U8	V4	X	U4	V0	U0

Y8

Packed Luma-Only, 8 bits per component. Every luma-only pixel in memory is represented with 8 bits, as shown. The images need be stored in memory in raster order, that is, top-left pixel first, bottom-right pixel last. Y8 is presented as YUV 4:4:4 on the AXI4-Stream interface.

7:0
Y

Y10

Packed Luma-Only, 10 bits per component. Every three luma-only pixels in memory is represented with 32 bits, as shown. The images need be stored in memory in raster order, that is, top-left pixel first, bottom-right pixel last. Y10 is presented as YUV 4:4:4 on the AXI4-Stream interface.

31:30	29:20	19:10	9:0
X	Y2	Y1	Y0

AXI4-Lite Control Interface

The AXI4-Lite interface allows you to dynamically control parameters within the core. The configuration can be accomplished using an AXI4-Lite master state machine, an embedded ARM®, or soft system processor such as MicroBlaze™. The Video Mixer can be controlled through the AXI4-Lite interface by using functions provided by the driver in the SDK. Another method is performing read and write transactions to the register space but should only be used when the first method is not available. Table 2-7 shows the AXI4-Lite control interface signals. This interface runs at the `ap_clk` clock.

Table 2-7: **AXI4-Lite Control Interface Signals**

Signal Name	I/O	Width	Description
s_axi_ctrl_aresetn	I	1	Reset
s_axi_ctrl_aclk	I	1	Clock
s_axi_ctrl_awaddr	I	18	Write Address
s_axi_ctrl_awprot	I	3	Write Address Protection
s_axi_ctrl_awvalid	I	1	Write Address Valid
s_axi_ctrl_awready	O	1	Write Address Ready
s_axi_ctrl_wdata	I	32	Write Data

Table 2-7: **AXI4-Lite Control Interface Signals *(Cont'd)***

Signal Name	I/O	Width	Description
s_axi_ctrl_wstrb	I	4	Write Data Strobe
s_axi_ctrl_wvalid	I	1	Write Data Valid
s_axi_ctrl_wready	O	1	Write Data Ready
s_axi_ctrl_bresp	O	2	Write Response
s_axi_ctrl_bvalid	O	1	Write Response Valid
s_axi_ctrl_bready	I	1	Write Response Ready
s_axi_ctrl_araddr	I	18	Read Address
s_axi_ctrl_arprot	I	3	Read Address Protection
s_axi_ctrl_arvalid	I	1	Read Address Valid
s_axi_ctrl_aready	O	1	Read Address Ready
s_axi_ctrl_rdata	O	32	Read Data
s_axi_ctrl_rresp	O	2	Read Data Response
s_axi_ctrl_rvalid	O	1	Read Data Valid
s_axi_ctrl_rready	I	1	Read Data Ready

Register Space

The Video Mixer has specific registers which allow you to dynamically control the operation of the core. All registers have an initial value of 0.

Top-Level Registers

Table 2-8 provides a detailed description of all the registers that apply globally to the IP.

Table 2-8: **Top-Level Registers**

Address (hex) BASEADDR+	Register Name	Access Type	Register Description
0x0000	Control	R/W	Bit[0] = ap_start Bit[1] = ap_done Bit[2] = ap_idle Bit[3] = ap_ready Bit[7] = auto_restart Others = Reserved
0x0004	Global Interrupt Enable	R/W	Bit[0] = Global interrupt enable Others = Reserved

Table 2-8: **Top-Level Registers *(Cont'd)***

Address (hex) BASEADDR+	Register Name	Access Type	Register Description
0x0008	IP Interrupt Enable		Bit[0] = Channel 0 (ap_done) Bit[1] = Channel 1 (ap_ready) Others = reserved
0x000C	IP Interrupt Status Register	R/TOW(1)	Bit[0] = Channel 0 (ap_done) Bit[1] = Channel 1 (ap_ready) Others = Reserved
0x0010	Width	R/W	Active width of background.
0x0018	Height	R/W	Active height of background.
0x0028	Background_Y_R	R/W	Red or Y value of background color
0x0030	Background_U_G	R/W	Green or U value of background color
0x0038	Background_V_B	R/W	Blue or V value of background color
0x0040	Layer enable	R/W	Bit[0] = Master layer is enabled/disabled Bit[1] = Layer 1 is enabled/disabled ... Bit[7] = Layer 7 is enabled/disabled Bit[8] = Logo layer is enable/disabled

Notes:
1. TOW = Toggle on Write.

Control (0x0000) Register

This register controls the operation of the Video Mixer. Bit[0] of the Control register, `ap_start`, kicks off the core from software. Writing 1 to this bit, starts the core to generate a video frame. To set the core in free running mode, Bit[7] of this register, `auto_restart`, must be set to 1. Bits[3:1] are not used now but reserved for future use.

Global Interrupt Enable (0x0004) Register

This register is the master control for all interrupts. Bit[0] can be used to enable/disable all core interrupts.

IP Interrupt Enable (0x0008) Register

This register allows interrupts to be enabled selectively. Currently, two interrupt sources are available `ap_done` and `ap_ready`. `ap_done` is triggered after the frame processing is complete, while `ap_ready` is triggered after the core is ready to start processing the next frame.

IP Interrupt Status (0x000C) Register

This is a dual purpose register. When an interrupt occurs, the corresponding interrupt source bit is set in this register. In readback mode (Get status), the interrupting source can be determined. In writeback mode (Clear interrupt), the requested interrupt source bit is cleared.

Width (0x0010) Register

The WIDTH register encodes the number of active pixels per scan. Supported values are between 64 and the value provided in the **Maximum Number of Columns** field in the Vivado Integrated Design Environment (IDE). To avoid processing errors, you should restrict values written to width to the range supported by the core instance. Furthermore, width needs to be a multiple of the **Samples per Clock** field in the Vivado IDE.

Height (0x0018) Register

The Height register encodes the number of active scan lines per frame. Supported values are between 64 and the value provided in the **Maximum Number of Rows** field in the Vivado IDE. To avoid processing errors, you should restrict values written to height to the range supported by the core instance.

Background_Y_R (0x0028) Register

Red color component of the background color. The range of the background color is determined by the number of bits per color component selected in the **Maximum Data Width** field in the Vivado IDE, for example, 0..1023 for 10-bit maximum data width.

Background_U_G (0x0030) Register

Green color component of the background color. The range of the background color is determined by the number of bits per color component selected in the **Maximum Data Width** field in the Vivado IDE, for example, 0..1023 for 10-bit maximum data width.

Background_V_B (0x0038) Register

Blue color component of the background color. The range of the background color is determined by the number of bits per color component selected in the **Maximum Data Width** field in the Vivado IDE, for example, 0..1023 for 10-bit maximum data width.

Layer Enable (0x0040) Register

This register has one bit for every layer that indicates whether a layer is enabled (1) or disabled (0). Bit[0] is for the master input layer, which associates with the `s_axis_video` AXI4-Stream input. This is the bottom-most layer, and all other layers are blended on top.

If this layer is disabled and no other layers are blended on top, the background color as specified by the previous mentioned registers are shown.

Bit[1] is for layer 1, which associates with either `s_axis_video1` AXI4-Stream input or `m_axi_mm_video1` memory mapped AXI4 input, depending on the **Interface Type** field selected in the Vivado IDE. Bit[2] is for layer 2, and so on, until Bit[7] which enables/disables layer 7.

Bit[8] is for the logo layer. The logo layer is the top-most layer and is blended on top of all other layers.

Layer Registers

Table 2-9 provides a detailed description of all the registers that apply to layers 1 through 7.

Table 2-9: **Layer Registers**

Address (hex) BASEADDR+	Register Name	Access Type	Description
0x0048+i*8	Layer i+1 Buffer	R/W	Start address of frame buffer for layer i+1. Only valid in case layer i+1 is a memory layer.
0x0088+i*8	Layer i+1 Alpha	R/W	Alpha blending value for layer i+1 ranging from 0 = Fully transparent 256 = Fully opaque
0x00C8+i*8	Layer i+1 Start X	R/W	X position of the top left corner of layer i+1, relative to the background layer
0x0108+i*8	Layer i+1 Start Y	R/W	Y position of the top left corner of layer i+1, relative to the background layer
0x0148+i*8	Layer i+1 Width	R/W	Active width (in pixels) of layer i+1
0x0188+i*8	Layer i+1 Height	R/W	Active height (in lines) of layer i+1
0x01c8+i*8	Layer i+1 Scale Factor	R/W	Scale factor for layer i+1 ranging from 0 = No scaling 1 = 2x scaling (horizontally and vertically) 2 = 4x scaling (horizontally and vertically)
0x0278+i*8	Layer i+1 Stride	R/W	Active stride (in bytes) of layer i+1

*Layer Buffer (0x0048+i*8) Register*

In case the layer is a memory layer, the layer buffer specifies the frame buffer address that holds the pixel data for this layer. Note that for the semi-planar formats (Y_UV8, Y_UV8_420, Y_UV10, and Y_UV10_420), it is assumed that the chroma buffer is offset by Layer Stride × Layer Height bytes from the luma frame buffer address. The address needs to be aligned to the data size of the memory mapped AXI4 interface. For the Video Mixer, the data size of the memory mapped AXI4 interface is 64 × Samples per Clock bits, that is, 64, 128, or 256 bits for 1, 2, and 4 samples per clock, respectively. This register is not applicable when the layer is a streaming layer.

*Layer Alpha (0x0088+i*8) Register*

The Layer Alpha register specifies the per layer alpha blending value used for blending this layer with the underlying layer. The value of this register has a range from 0 which is fully transparent, to 256 which is fully opaque. Layer alpha blending is only supported when the **Enable Global Alpha** field in the Vivado IDE is enabled for a particular layer. When alpha blending is disabled for a layer, layers are blended as fully opaque on top of the underlying layer.

*Layer Start X (0x00c8+i*8) Register*

The Layer Start X register marks the x position (columns) of the top left corner of the layer relative to the mixer output frame dimensions. (0,0) puts the layer in the top left corner of the output frame. To avoid processing errors, you should restrict values written to start x such that the entire layer is contained within the frame. Furthermore, the x position needs to be a multiple of the **Samples per Clock** field in the Vivado IDE.

*Layer Start Y (0x0108+i*8) Register*

The Layer Start Y register marks the y position (rows) of the top left corner of the layer relative to the mixer output frame dimensions. (0,0) puts the layer in the top left corner of the output frame. To avoid processing errors, you should restrict values written to start y such that the entire layer is contained within the frame.

*Layer Width (0x0148+i*8) Register*

The Layer Width register encodes the active width in pixels of the layer. Supported values are between 64 and the value provided in the **Maximum Number of Columns** field in the Vivado IDE if the **Enable Scaling** field in the Vivado IDE is disabled. It is between 64 and the **Layer Line Buffer Width** in the Vivado IDE if the **Enable Scaling** field is enabled. To avoid processing errors, you should restrict values written to layer width such that the entire layer is contained within the frame. Furthermore, layer width needs to be a multiple of the **Samples Per Clock** field in the Vivado IDE.

Layer Height (0x0188+i*8) Register

The Layer Height register encodes the height in lines of the layer. Supported values are between 64 and the value provided in the **Maximum Number of Rows** field in the Vivado IDE. To avoid processing errors, you should restrict values written to layer height such that the entire layer, after applying the scale factor, is contained within the frame.

Layer Scale Factor (0x01C8+i*8) Register

The Layer Scale Factor register determines the scaling factor that is applied before blending this layer with the underlying layer. Scale factors of 1x, 2x, and 4x are supported.

- 0 = No scaling
- 1 = 2x scaling (horizontally and vertically)
- 2 = 4x scaling (horizontally and vertically)

Layer scaling is only supported when the **Enable Scaling** field in the Vivado IDE is enabled for a particular layer.

Layer Stride (0x0278+i*8) Register

This register is not applicable if the layer is a streaming layer. If it is a memory layer, the layer stride determines the number of bytes from one row of pixels in memory to the next row of pixels in memory. When a video frame is stored in memory, the memory buffer might contain extra padding bytes after each row of pixels. The padding bytes only affect how the image is stored in memory, but does not affect how the image is displayed.

Padding bytes is necessary to make sure that every row of pixels starts at an address that is aligned with the size of the data on the memory mapped AXI4 interface. Therefore, layer stride needs to be a multiple of the memory mapped AXI4 data size. For the Video Mixer, the data size of the memory mapped AXI4 interface is 64 × Samples per Clock bits, that is, 64, 128, or 256 bits for 1, 2, and 4 samples per clock, respectively.

Logo Layer Registers

Table 2-10 provides a detailed description of all the registers that apply to the logo layer.

Table 2-10: **Logo Layer Registers**

Address (hex) BASEADDR+	Register Name	Access Type	Description
0x0240	Logo Start X	R/W	X position of the top left corner of the logo, relative to the output resolution
0x0248	Logo Start Y	R/W	Y position of the top left corner of the logo, relative to the output resolution
0x0250	Logo Width	R/W	Active width (in pixels) of the logo

Table 2-10: **Logo Layer Registers *(Cont'd)***

Address (hex) BASEADDR+	Register Name	Access Type	Description
0x0258	Logo Height	R/W	Active height (in lines) of the logo
0x0260	Logo Scale Factor	R/W	Scale factor for logo ranging from 0 = No scaling 1 = 2x scaling (horizontally and vertically) 2 = 4x scaling (horizontally and vertically)
0x0268	Logo Alpha	R/W	Alpha blending value for logo ranging from 0 = Fully transparent 255 = Fully opaque
0x02B0	Logo Color Key Min R	R/W	Red minimum value of color key range
0x02B8	Logo Color Key Min G	R/W	Green minimum value of color key range
0x02C0	Logo Color Key Min B	R/W	Blue minimum value of color key range
0x02C8	Logo Color Key Max R	R/W	Red maximum value of color key range
0x02D0	Logo Color Key Max G	R/W	Green maximum value of color key range
0x02D8	Logo Color Key Max B	R/W	Blue maximum value of color key range
0x1 0000	Logo Red Buffer	R/W	Start address of buffer for red logo pixels
0x2 0000	Logo Green Buffer	R/W	Start address of buffer for green logo pixels
0x3 0000	Logo Blue Buffer	R/W	Start address of buffer for blue logo pixels
0x4 0000	Logo Alpha Buffer	R/W	Start address of buffer for logo per pixel alpha

Logo Start X (0x0240) Register

The Logo Start X register marks the x position (columns) of top left corner of the logo relative to the mixer output frame dimensions. (0,0) puts the logo in the top left corner of the output frame. To avoid processing errors, you should restrict values written to start x such that the entire logo is contained within the frame. Furthermore, the x position needs to be a multiple of the **Samples per Clock** field in the Vivado IDE.

Logo Start Y (0x0248) Register

The Logo Start Y register marks the y position (rows) of the top left corner of the logo relative to the mixer output frame dimensions. (0,0) puts the logo in the top left corner of the output frame. To avoid processing errors, you should restrict values written to start y such that the entire logo is contained within the frame.

Logo Width (0x0250) Register

The Logo Width register encodes the width in pixels of the logo. Supported values are between 32 and the value provided in the **Maximum Number of Columns for Logo** field in the Vivado IDE. To avoid processing errors, you should restrict values written to width to

the range supported by the core instance. Furthermore, width needs to be a multiple of the **Samples per Clock** field in the Vivado IDE.

Logo Height (0x0258) Register

The Logo Height register encodes the height in lines of the logo. Supported values are between 32 and the value provided in the **Maximum Number of Rows for Logo** field in the Vivado IDE. To avoid processing errors, you should restrict values written to height to the range supported by the core instance.

Logo Scale Factor (0x0260) Register

The Logo Scale Factor register determines the scaling factor that is applied before blending the logo with the underlying layer. Scale factors of 1x, 2x, and 4x are supported.

- 0 = No scaling
- 1 = 2x scaling (horizontally and vertically)
- 2 = 4x scaling (horizontally and vertically)

Logo Alpha (0x0268) Register

The Logo Alpha register specifies the per layer alpha blending value used for blending the logo with the underlying layer. The value of this register has a range from 0 which is fully transparent, to 256 which is fully opaque.

Logo Color Key Min R (0x02B0) Register

The Logo Color Key values determine a color range that is treated as transparent. Any logo pixel that falls in this range are not blended on top of the underlying layer but is transparent. This equation is used to determine if a pixel is transparent.

Let (r, g, b) be a pixel value of the logo.

Let (min_r, min_g, min_b) and (max_r, max_g, max_b) be the values of the Color Key registers.

Then, pixel (r, g, b) is transparent if the following holds:

$$min\_r \leq r \leq max\_r \text{ AND}$$

$$min\_g \leq g \leq max\_g \text{ AND}$$

$$min\_b \leq b \leq max\_b$$

To turn off background color keying, program a maximum value that is smaller than the minimum value, for example, maximum is 0, minimum is 1.

Logo Color Key Min G (0x02B8)

For more information, see Logo Color Key Min R (0x02B0) Register.

Register, Logo Color Key Min B (0x02C0) Register

For more information, see Logo Color Key Min R (0x02B0) Register.

Logo Color Key Max R (0x02C8) Register

For more information, see Logo Color Key Min R (0x02B0) Register.

Logo Color Key Max G (0x02D0) Register

For more information, see Logo Color Key Min R (0x02B0) Register.

Logo Color Key Max B (0x02D8) Register

For more information, see Logo Color Key Min R (0x02B0) Register.

Logo Red Buffer (0x1 0000) Register

The memory for storing the frame data for the logo layer is block RAM local to the core. The application is required to load the logo into this block RAM through the AXI4-Lite interface. The Logo Buffer address is the start address for this block RAM.

The Logo Buffer addresses point to block RAM for separate red, green, and blue pixel buffers that hold the logo pixels. Optionally, if Logo Per Pixel Alpha is enabled, there is an additional buffer for the logo per pixel alpha values. The logo has to be formatted as planar RGB(A) with eight bits per color component. The size of the buffer is dimensioned by the **Maximum Number of Columns for Logo** and the **Maximum Number of Rows for Logo** fields in the Vivado IDE. With eight bits per color component, the size in bytes is defined as columns × rows. The logo pixels have to be organized in the buffer as follows.

- Red[x + y × columns] holds the red pixel of the logo for column x and row y
- Green[x + y × columns] holds the green pixel of the logo for column x and row y
- Blue[x + y × columns] holds the blue pixel of the logo for column x and row y
- Alpha[x + y × columns] holds the per pixel alpha value of the logo for column x and row y

Logo Green Buffer (0x2 0000) Register

For more information, see Logo Red Buffer (0x1 0000) Register.

Logo Blue Buffer (0x3 0000) Register

For more information, see Logo Red Buffer (0x1 0000) Register.

Logo Alpha Buffer (0x4 0000) Register

For more information, see Logo Red Buffer (0x1 0000) Register.

Designing with the Core

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

General Design Guidelines

The Video Mixer has support for up to eight layers, with an optional logo layer, using a combination of video inputs from either frame buffer (through memory-mapped AXI4 interfaces) or streaming video cores (through AXI4-Stream interfaces). Figure 3-1 shows the functional block diagram of the Video Mixer. Functions listed as optional are under Vivado® Integrated Design Environment (IDE) control, as explained later in this section.

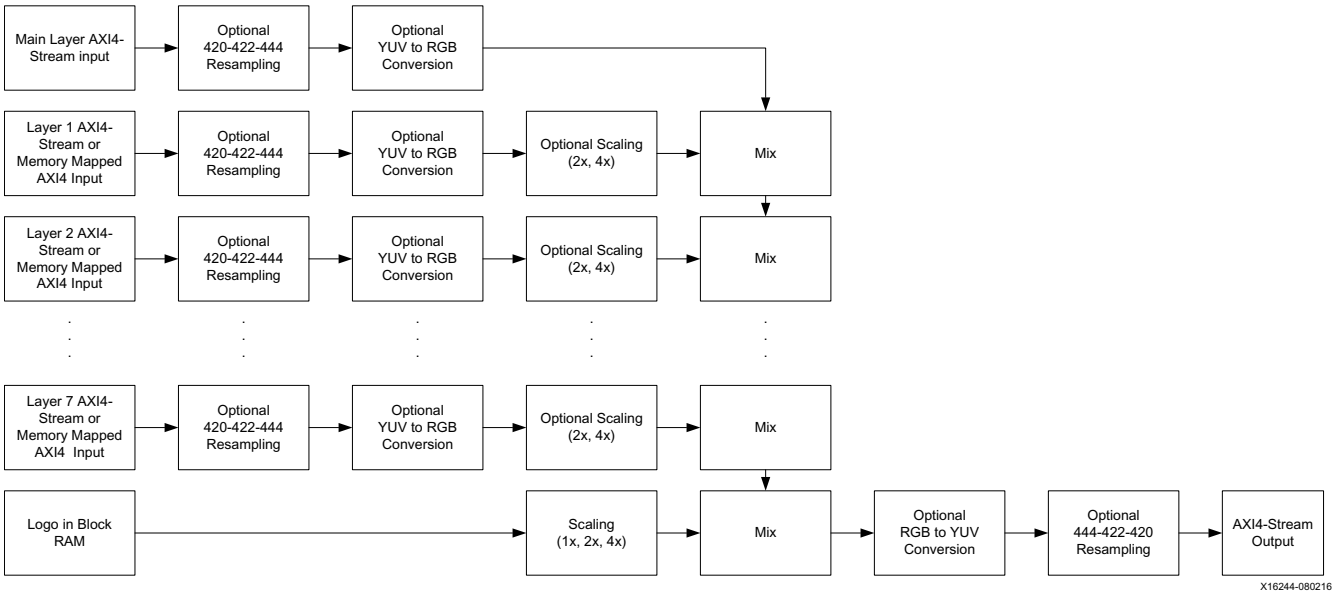

Figure 3-1: **Video Mixer Customize IP**

The Video Mixer always has one streaming input layer and one streaming output layer. This is known as main, master, or background layer. The main layer associates with the `s_axis_video` AXI4-Stream input and `m_axis_video` AXI4-Stream output.

The main layer input is the bottom-most layer and all other layers are blended on top. Note that there is no programmable Z order for layers. The main layer can be disabled at run-time through the AXI4-Lite interface, by clearing Bit[0] of the Layer Enable (0x0040) Register. When disabled, the main layer does not read from the streaming interface but generates a solid background color as specified by Background_Y_R (0x0028) Register, Background_U_G (0x0030) Register, and Background_V_B (0x0038) Register.

The video format of both the streaming input and output layer is determined by the **Video Format** field in the Vivado IDE. As video mixing is performed in the RGB domain, note that if the selected video format is YUV 4:4:4, color space conversion is done at the input to RGB, and at the output to go back to YUV 4:4:4 again. If the selected video format is YUV 4:2:2 or YUV 4:2:0, additional chroma resampling at the input and output is performed to go from YUV 4:2:2 (or YUV 4:2:0) to YUV 4:4:4 and back.

The Video Mixer can have up to seven additional video or graphics sources, each of which can be configured to be an AXI4-Stream or memory mapped AXI4 frame buffer. When using streaming layers, keep in mind that the Video Mixer achieves synchronization of streaming layers to the main output layer by throttling the incoming data if the mixer is not ready to accept this data yet. Also, the mixer has no internal buffering to queue up incoming data.

If a memory mapped AXI4 frame buffer is selected to be a source, then the Video Mixer automatically handles interfacing to memory without the need of an additional AXI VDMA controller.

Like the main layer, every layer has a pre-defined video format that is either RGB, YUV 4:4:4, YUV 4:2:2, or YUV 4:2:0. Optional chroma resampling from YUV 4:2:2, or YUV 4:2:0 to YUV 4:4:4, and optional color space conversion from YUV 4:4:4 to RGB, are performed as mixing is done in the RGB domain. Layers 1 through 7, additionally can be configured to optionally have scaling ability (1x, 2x, or 4x) by the **Enable Scaling** field in the Vivado IDE. Scaling is implemented by means of pixel and line repeat, and therefore requires an internal line buffer.

The Video Mixer provides an optional logo layer. It blends a logo that is stored in the block RAM on the top-most layer. A programmable color key can be used to make part of the logo transparent. Also, (per pixel) alpha-blending can be used for logo transparency. The logo layer also has the ability for scaling (1x, 2x, or 4x). As the logo is read from the block RAM, no additional line buffer is needed for this. Scaling is implemented by means of pixel and line repeat.

Alpha Blending

Alpha blending is the process of combining two images with the appearance of partial transparency. To perform this composition, a per layer alpha value (and optionally a per pixel alpha value) is used that contains the coverage information for all the pixels within a layer. The alpha value ranges from 0 to 1, where 0 represents that the current pixel does not contribute to the final image and is fully transparent. A 1 represents that the current pixel is fully opaque. Any value in between represents a partially transparent pixel.

When applying alpha blending, the two pixels to be blended reside within two different image layers. Each layer has a definite Z-axis 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-axis is below. This operation is conceptually simple linear interpolation between each color component of each layer.

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

Where:

α is the product of the global alpha and per pixel alpha (if enabled) or just the global alpha value (otherwise).

$Component_{(x, y, z)}$ 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 applies for the next layer above, Layer z + 1. These alpha-blending operations can be chained together 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 mode, any number of image layers can be blended by taking the blended result of the layer below it.

Note: Alpha blending is optional per layer but always enabled for the logo layer. In case alpha blending is disabled, pixels are always superimposed as fully opaque on the underlying layer.

Clocking

The Video Mixer has only one clock domain. All interfaces, that is, master and slave AXI4-Stream video interfaces as well as the AXI4-Lite interface, and also the memory mapped AXI4 interfaces uses the `ap_clk` pin as its clock source.

Pixel throughput of the Video Mixer core is defined by the product of the clock frequency times the **Samples per Clock** setting in the Vivado IDE. With a clock frequency of 300 MHz for `ap_clk`, and a two sample per clock configuration, the Video Mixer is capable of a 600 mega pixel throughput rate, which is sufficient to handle 4K resolutions at 60Hz.

Resets

The Video Mixer has only a hardware reset option, `ap_rst_n` pin. No software reset option is available. The external reset pulse needs to be held for 16 or more `ap_clk` cycles to reset the core. The `ap_rst_n` signal is synchronous to the `ap_clk` clock domain. The `ap_rst_n` signal resets the entire core including the AXI4-Lite, AXI4-Stream, and memory mapped AXI4 interfaces.

System Considerations

The Video Mixer must be configured for the actual input and output image frame-size to operate properly. To gather the frame size information from the image video stream, it can be connected to the Video In to AXI4-Stream input and the Video Timing Controller. The timing detector logic in the Video Timing Controller gathers the video timing signals. The AXI4-Lite control interface on the Video Timing Controller allows the system processor to read out the measured frame dimensions, and program all downstream cores, such as the Video Mixer, with the appropriate image dimensions.

Another system consideration that needs to be taken into account is that there is sufficient bandwidth available to the Video Mixer to maintain proper functioning memory layers. The bandwidth needed (in MB/s) for a memory layer can be calculated with the following equation:

$$Bandwidth\ (MB/s) = fps \times height \times stride$$

Where *fps* is the number of frames per second the Video Mixer is operating at, *height* is the height in lines of the layer, and *stride* is the stride in bytes of the layer.

Programming Sequence

Most Video Mixer processing parameters other than image sizes can be changed dynamically and the change is picked up immediately. If the image size needs to be changed or the entire system needs to be restarted, it is recommended that pipelined Xilinx IP video cores are disabled/reset from system output towards the system input, and programmed/enabled from system output to system input.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 2]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 2] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the Vivado IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4].

Note: Figure in this chapter is an illustration of the Vivado Integrated Design Environment (IDE). The layout depicted here might vary from the current version.

Interface

The Video Mixer is configured to meet your specific needs through the Vivado Design Suite. This section provides a quick reference to parameters that can be configured at generation time.

Figure 4-1 shows the Video Mixer Vivado IDE main configuration screen.

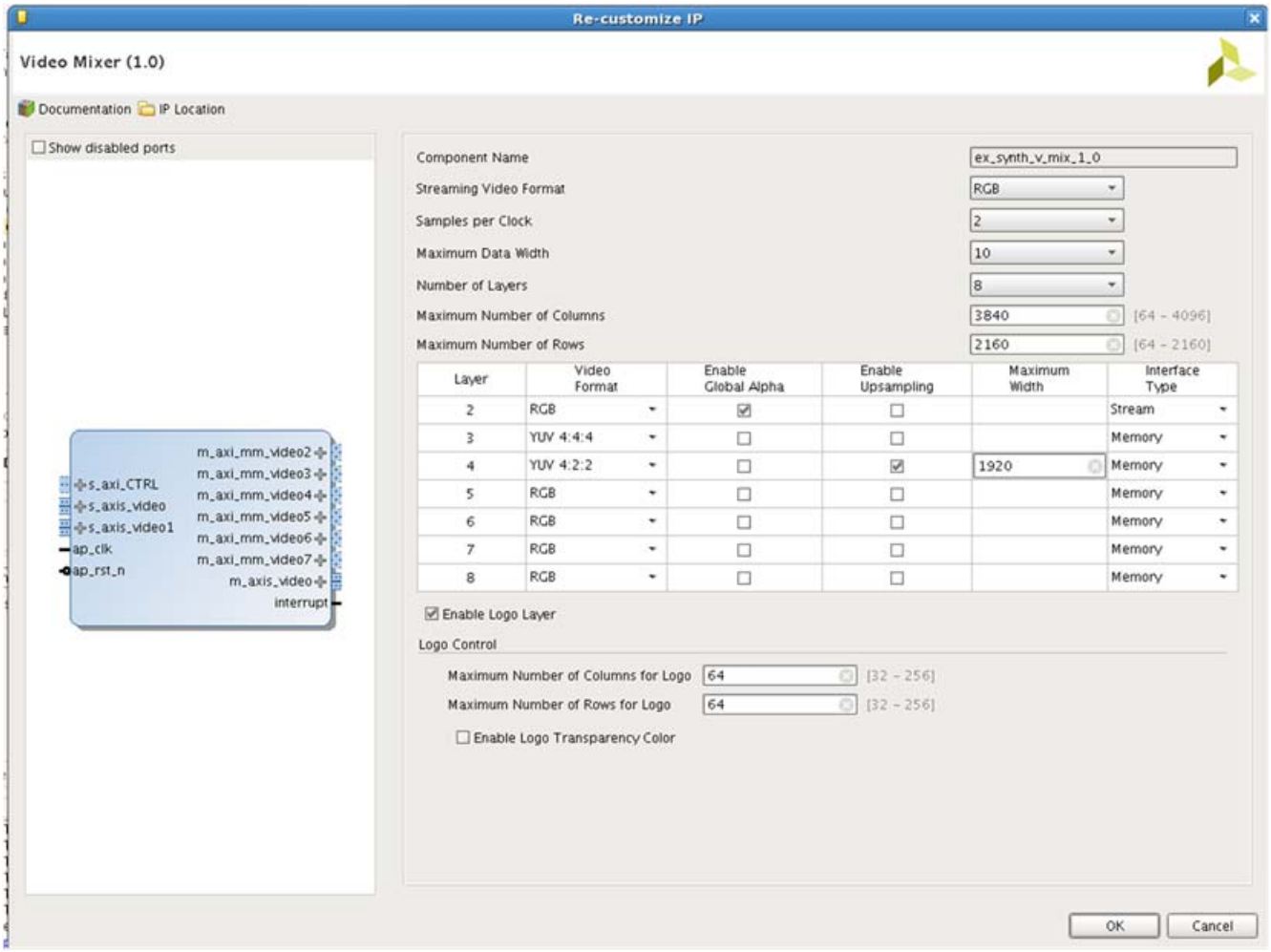

Figure 4-1: **Video Mixer Customize IP**

General Settings

The following settings are generally applicable:

- **Component Name** – The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_".
- **Streaming Video Format** – Specifies the video format on the streaming input (`s_axis_video`) and output (`m_axis_video`) AXI4-Stream interfaces. Possible formats are RGB, YUV 4:4:4, YUV 4:2:2, or YUV 4:2:0.

  Note: The video format is chosen at build time and cannot be changed at run-time.
- **Samples Per Clock** – Specifies the number of pixels processed per clock cycle. Permitted values are one, two, and four samples per clock. This parameter determines the IP throughput. The more samples per clock, the larger throughput it provides. The larger throughput always needs more hardware resources.

  Note: This property applies to all layers that have a streaming interface.
- **Maximum Data Width** – Specifies the bit width of input and output samples on all the streaming interfaces. Permitted values are 8, 10, 12, and 16 bits.

  Note: This property applies to all layers that have a streaming interface.
- **Maximum Number of Columns** – Specifies maximum active video columns/pixels the IP core could produce at run-time. Any video width that is less than the **Maximum Number of Columns** can be programmed through AXI4-Lite control interface without regenerating the core.
- **Maximum Number of Rows** – Specifies maximum active video rows/lines the IP core could produce at run-time. Any video height that is less than **Maximum Number of Rows** can be programmed through the AXI4-Lite control interface without regenerating the core.
- **Number of Layers** – Specifies the number of layers with a minimum of one and a maximum of eight. When only one layer is selected, the logo layer is enabled by default, and the Video Mixer is essentially just a logo overlay function.

Layer Settings

The following settings apply to optional layers with ID 1 through 7.

- **Layer ID** – Identifies the layer ID.
- **Video Format** – Specifies the video format per layer. Possible formats are RGB, YUV 4:4:4, YUV 4:2:2, or YUV 4:2:0 in case the layer interface type is streaming, and RGBX8, YUVX8, RGBA8, BGRA8, YUVA8, YUYV8, RGBX10, YUVX10, RGB565, Y_UV8, and Y_UV8_420, RGB8, YUV8,Y_UV10, Y_UV10_420, Y8, Y10, otherwise.

  Note: The video format is chosen at build time and cannot be changed at run-time.

- **Enable Global Alpha** – When selected, this enables alpha blending for this layer with the layer underneath. The alpha value needs to be programmed at run-time through the AXI4-Lite control interface.

  Note: If either the RGBA8, BRGA8, or YUVA8 per pixel alpha formats are selected for a layer, then the global alpha is automatically enabled.

- **Enable Scaling** – When selected, this enables scaling for this layer. The scale factor (1x, 2x, or 4x) needs to be programmed at run-time through the AXI4-Lite control interface.
- **Line Buffer Width** – This is only enabled when scaling is enabled for a layer and specifies the maximum width of this layer before scaling. A line buffer with this dimension is allocated in the block RAM to allow for scaling through line repeat. The width needs to be a multiple of the samples per clock setting.
- **Interface Type** – Determines whether a layer is either a memory layer or a streaming layer. A memory layer has a memory mapped AXI4 interface while a streaming layer has an AXI4-Stream interface.

Logo Layer Settings

The following settings apply to the optional logo layer.

- **Enable Logo Layer** – When selected, this includes the logo layer. Block RAM is used for pixel storage of the logo according to the dimension settings. The logo is limited to eight bits per color component, and always needs to be formatted as RGB.
- **Maximum Number of Columns for Logo** – Specifies maximum pixel width of the logo. The width needs to be a multiple of the samples per clock setting. This setting affects the block RAM utilization.
- **Maximum Number of Rows for Logo** – Specifies maximum line height of the logo. This setting affects the block RAM utilization.
- **Enable Logo Transparency Color** – When selected, this allows for run-time programmability of a specified color key that becomes transparent.
- **Enable Logo Per Pixel Alpha** – When selected, this adds per pixel alpha blending functionality for the logo.

User Parameters

Table 4-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Top-Level Parameters		
Number of Video Components	NUM_VIDEO_COMPONENTS	3
Samples per Clock	SAMPLES_PER_CLOCK	1
Maximum Data Width	MAX_DATA_WIDTH	8
Maximum Number of Columns	MAX_COLS	3,840
Maximum Number of Rows	MAX_ROWS	2,160
Video Format	VIDEO_FORMAT	RGB
RGB	0	
YUV 4:4:4	1	
YUV 4:2:2	2	
YUV 4:2:0	3	
Number of Layers	NR_LAYERS	4
Layer Parameters		
Layer *i* Video Format	LAYER*i*_VIDEO_FORMAT	RGB
RGB	0	
YUV 4:4:4	1	
YUV 4:2:2	2	
YUV 4:2:0	3	
RGBX8	10	
YUVX8	11	
YUYV8	12	
RGBA8	13	
YUVA8	14	
RGBX10	15	
YUVX10	16	
RGB565	17	
Y_UV8	18	
Y_UV8_420	19	
RGB8	20	
YUV8	21	
Y_UV10	22	

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Y_UV10_420	23	
Y8	24	
Y10	25	
BGRA8	26	
Layer *i* Enable Global Alpha	LAYER*i*_ALPHA	FALSE
Layer *i* Enable Scaling	LAYER*i*_UPSAMPLE	FALSE
Layer *i* Maximum Width	LAYER*i*_MAX_WIDTH	1,920
Layer *i* Interface Type	LAYER*i*_INTF_TYPE	Memory
Memory	0	
Stream	1	
Logo Parameters		
Enable Logo Layer	LOGO_LAYER	FALSE
Maximum Number of Columns for Logo	MAX_LOGO_COLS	64
Maximum Number of Rows for Logo	MAX_LOGO_ROWS	64
Enable Logo Transparency Color	LOGO_TRANSPARENCY_COLOR	FALSE
Enable Logo per Pixel Alpha	LOGO_PIXEL_ALPHA	FALSE

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5].

IMPORTANT: *For cores targeting 7 series or Zynq-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

Example Design

This chapter provides two example systems that include the Video Mixer. One is a simulation example design and the other one is a synthesizable example design. Important system-level aspects when designing with the Video Mixer are highlighted in these example designs, including:

- Video Mixer usage with memory mapped AXI4 interface memory layers.
- Typical usage of the Video Mixer in conjunction with other cores.
- Run-time configuration of the Video Mixer by programming registers on-the-fly.

Note: The synthesizable example design is only available on the Xilinx® KC705 evaluation board.

To open the example project, perform the following:

1. Select the **Video Mixer IP** from the Vivado® IP catalog.
2. Double-click the selected IP or right-click the IP and select **Customize IP** from the menu.
3. Configure the build-time parameters in the **Customize IP** window and click **OK**. The Vivado IDE generates an example design matching the build-time configuration.
4. In the **Generate Output Products** window, select **Generate** or **Skip**. If **Generate** is selected, the IP output products are generated after a brief moment.
5. Right-click **Video Mixer** in **Sources** panel and select **Open IP Example Design** from the menu.
6. In the **Open IP Example Design** window, select example project directory and click **OK**. The Vivado software then runs automation to generate the example design in the selected directory.

The generated project contains two example designs. Figure 5-1 shows the **Source** panel of the example project. Synthesizable example block design, along with top-level file, resides in **Design Sources** catalog. A corresponding constraint file is also provided for the synthesizable example design. Simulation example design files (including block design file, SystemVerilog test bench and another task file) are under **Simulation Sources**.

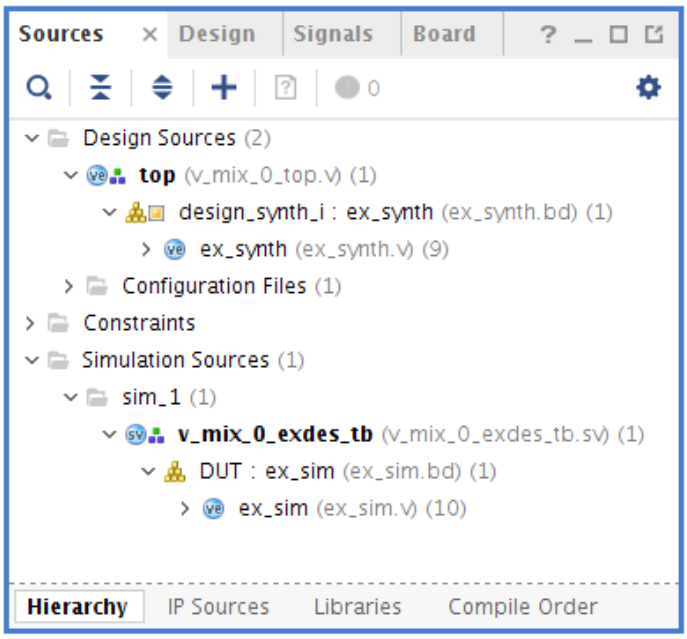

Figure 5-1: **Video Mixer Example Project Source Panel**

Simulation Example Design

The simulation example design contains the following video IP cores: Video Test Pattern Generator, Video Mixer, Video Timing Controller, and the AXI4-Stream to Video Output bridge. The design also contains a AXI Verification IP (VIP) core (to enable register programming) connected to an AXI interconnect. An AXI block RAM (BRAM) controller with associated block RAM memory is also connected to this AXI interconnect.

Note: The simulation example design currently has limited configurability as it is fixed to have two layers: the main streaming layer, and one additional memory layer.

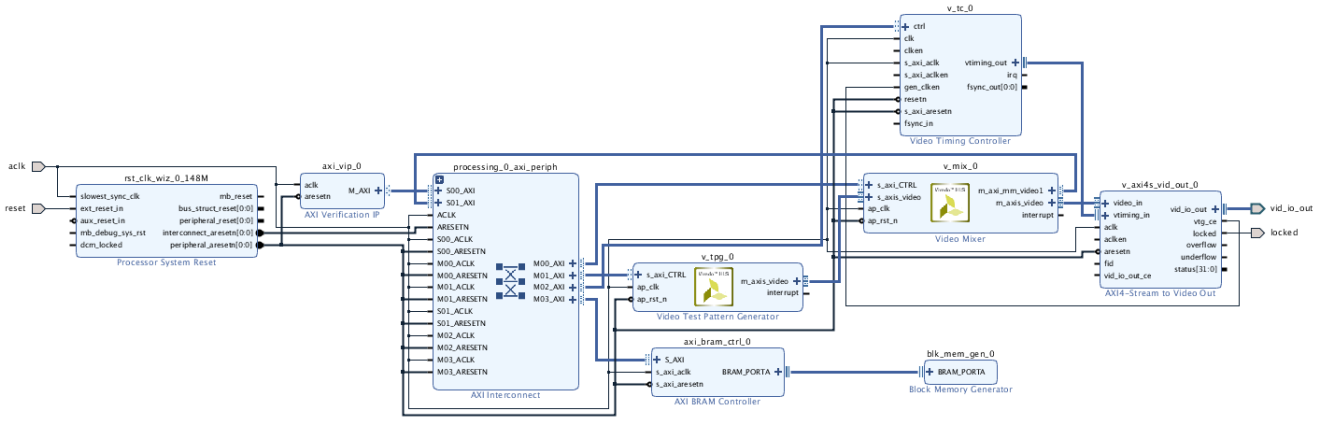

Figure 5-2: **Video Mixer Simulation Design**

AXI VIP acts as AXI master in the system to drive the TPG, Video Mixer, and Video Timing Controller cores. It configures width, height, pattern, and other registers of the TPG core. Next, the layer properties of the Video Mixer are programmed. Then, timing parameters of the Video Timing Controller core are configured by the AXI VIP core.

After all configurations are performed, the AXI VIP core starts the TPG, Video Mixer, and Video Timing Controller cores. Because this design runs an RTL simulation, running large video frames can take a long time. Xilinx recommends running a small video size for this example design, by default a 64 × 64 frame dimension is being programmed. Width and height values, as well as other register settings, can be changed in the simulation test bench `v_mix_0_exdes_tb.sv` file.

The `v_mix_0` core is in free-running mode after kickoff. It generates video stream pixels at a clock rate of `ap_clk`.

The `v_mix_0` core receives video frames through the AXI4 stream interface of the master layer, and additionally reads frames from a memory mapped AXI4 memory layer, that are then mixed and sent out over the AXI4-Stream master interface.

The AXI4-Stream to Video Out core, working with the Video Timing Controller, interfaces with the AXI4-Stream interface implementing a timed Video Protocol to a video source (parallel video data with video syncs and blanks).

The simulation example design checks the output port named `locked` from the AXI4-Stream to Video Out core. The `locked` port indicates that the output timing is locked to the output video. The simulation example design indicates that the test completed successfully if video lock is successfully detected.

Synthesizable Example Design

The difference between the Synthesizable design and the Simulation example design is the use of the MicroBlaze™ microprocessor instead of the AXI VIP core as AXI4 master. Also, the synthesizable design uses the MIG IP core for DDR memory access. The `locked` port of AXI4-Stream to Video Out is connected to `axi_gpio_lock` core and the MicroBlaze polls the corresponding register for a sign that the test passed.

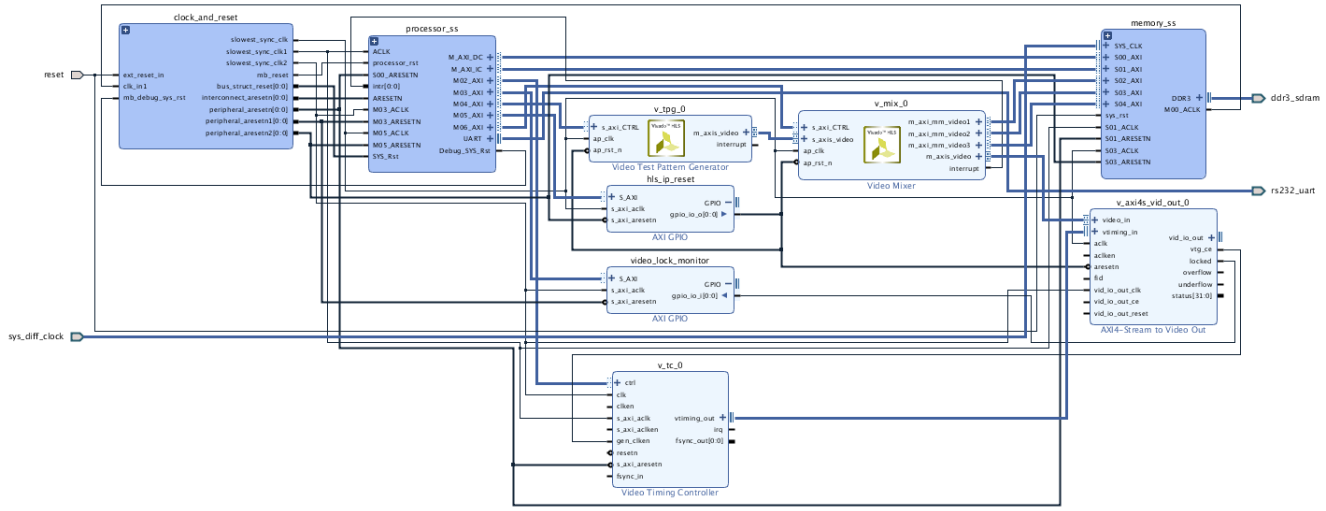

Figure 5-3: **Synthesizable Example Block Design**

The synthesizable example design requires both Vivado and Xilinx SDK tools.

The first step is to run synthesis, implementation and bitstream generation in Vivado. After all those steps are done, select **File > Export > Export Hardware**. In the window, select **Include bitstream**, select an export directory and click **OK**.

The remaining work is performed in the Xilinx SDK tool. The Video Mixer example design file can be found in the following SDK directory:

```
(/data/embeddedsw/XilinxProcessorIPLib/drivers/v_mix_v1_0/examples/
```

The example application design source files (contained within `examples` folder) are tightly coupled with the `v_mix` example design available in Vivado IP catalog.

`vmix_example.tcl` automates the process of generating the downloadable bit and `elf` files from the provided example `hdf` file.

To run the provided Tcl script:

1. Copy the exported example design `hdf` file in the `examples` directory of the driver
2. Launch the Xilinx Software Command-Line Tool (xsct) terminal
3. `cd` into the examples directory
4. Source the `tcl` file `xsct`:

   ```
   %>source vmix_example.tcl
   ```

5. Execute the script:

   ```
   xsct%>vmix_example <hdf_file_name.hdf>
   ```

The Tcl script performs the following:

- Create workspace
- Create HW project
- Create BSP
- Create Application Project
- Build BSP and Application Project

After the process is complete, the required files are available in:

```
bit file -> vmix_example.sdk/vmix_example_hw_platform folder
elf file -> vmix_example.sdk/vmix_example_design/{Debug/Release} folder
```

Next, perform the following steps to run the software application:

IMPORTANT: *To do so, make sure that the hardware is powered on and a Digilent Cable or an USB Platform Cable is connected to the host PC. Also, ensure that a USB cable is connected to the UART port of the KC705 board.*

1. Launch SDK.
2. Set workspace to `vmix_example.sdk` folder in prompted window. The SDK project opens automatically (if a welcome page shows up, close that page).
3. Download the bitstream into the FPGA by selecting **Xilinx Tools > Program FPGA**. The **Program FPGA** dialog box opens.
4. Ensure that the Bitstream field shows the bitstream file generated by Tcl script, and then click **Program**.

   Note: The DONE LED on the board turns green if the programming is successful.
5. A terminal program (HyperTerminal or PuTTY) is needed for UART communication. Open the program, choose appropriate port, set baud rate to 9,600 and establish Serial port connection.
6. Select and right-click the application `vmix_example_design` in `Project_Explorer` panel.
7. Select **Run As > Launch on Hardware (GDB)**.
8. Select **Binaries and Qualifier** in window and click **OK**.

   The example design test result are shown in terminal program.

For more information, visit www.xilinx.com/tools/sdk.htm.

When executed on the board, the operations are listed in `readme.text` in the examples folder. Video input tested are 1080p and 720p.

Test Bench

There is no test bench for this IP core release.

Verification, Compliance, and Interoperability

This appendix provides details about how this IP core was tested for compliance with the protocol to which it was designed.

Simulation

A highly parameterizable test bench was used to test the Video Mixer in Vivado® High-Level Synthesis (HLS). Testing included the following:

- Register accesses
- Processing multiple frames of data
- Varying IP throughput and pixel data width
- Testing the Video Mixer with AXI4-Stream and memory mapped AXI4 interface layers
- Testing of various frame sizes
- Varying parameter settings

Hardware Testing

The Video Mixer core has been validated at Xilinx® to represent many different parameterizations. A test design was developed for the core that incorporated a MicroBlaze™ processor, AXI4-Lite interconnect, and various other peripherals. The MicroBlaze processor was responsible for:

- Programing the video clock to match tested video resolution
- Configuring the video IP cores with different resolutions
- Launching the test
- Reporting the Pass/Fail status of the test and any errors that were found

Interoperability

The core slave (input) and master (output) AXI4-Stream interface can work directly with any core that produces RGB, YUV 4:4:4, YUV 4:2:2, or YUV 4:2:0 video data.

Migrating and Upgrading

This appendix contains information about upgrading to a more recent version of the IP core.

Upgrading in the Vivado Design Suite

This section is not applicable for the first release of the core.

Application Software Development

The Video Mixer core is delivered with a bare-metal driver as part of the SDK installation. The driver follows a layered architecture wherein layer 1 provides basic register peek/poke capabilities and requires you to be familiar with the register map and inner workings of the core. Layer 2, on the other hand, abstracts away all the lower level details and provides an easy to use functional interface to the Video Mixer. Xilinx® recommends always using layer 2 APIs to interact with the core.

Building the BSP

When the Board Support Package (BSP) is built, the Video Mixer driver inherently pulls in the required dependency, that is, the video common driver. This driver contains the enumerations for video specific information like color format, color depth, frame rate, etc. It also defines fundamental data types to represent the video stream, timing and window that are used in the Video Mixer driver and is common across all Xilinx video drivers.

During the build process the Video Mixer driver extracts the Video Mixer hardware configuration settings from the provided hardware design file.

Prerequisites

If optional layers are enabled and configured as "Memory," there are certain requirements that must be taken care of while programming the core.

1. The core itself does not have a data realignment engine and therefore the application software must align the layer memory addresses before writing to the registers. The alignment requirement is specified below, and makes sure that the start address is aligned with the width of the memory interface.

   2 × Pixels per Clock × 4 Bytes

2. When setting up the memory layer window, also the stride (in bytes) must be aligned as above to make sure that every row of pixels starts at an aligned memory location. To compute the stride from a window width in pixels the following equation can be used.

   $$\textit{Stride in Bytes} \geq (\textit{Width} \times \textit{Bytes per Pixel});$$

   Bytes per pixel varies per video in memory format, and is described in the section detailing the memory mapped AXI4 interface. Note that padding bytes are sometimes necessary (hence the ≥ in the equation) to make sure that every row of pixels starts at an address that is aligned with the size of the data on the memory mapped interface.

3. **Layer Window Start Horizontal Position and Width** must be a multiple of **Pixels per Clock** as selected in the Vivado Integrated Design Environment (IDE) for this core.

Modes of Operation

The Video Mixer supports two modes of operation, which require two different programming models.

- **Auto Restart Mode (Default)** – The driver initialization routine configures the Video Mixer for auto restart mode. In this mode, after the current frame is processed the core automatically triggers the start of the next frame processing. Consequently, the core can keep on processing frames without any software intervention, with settings applied when the core was started. This mode is used when there is no need to make frame synchronous changes to the core registers. This is the case if streaming layers are being used.

  You can switch to Auto Restart Mode, at any time, by disabling the interrupts, using the `XVMix_InterruptDisable` API.

- **Interrupt Mode** – When using memory layers, frame synchronous changes to the layer memory buffer address are key for the correct operation of the Video Mixer, that is, reading from memory needs to be synchronized with writing to memory by for example a graphics engine. In this case, the interrupt mode should be used. In this mode, the interrupt (IRQ) port of the core needs to be connected to a system interrupt controller.
  When an interrupt is triggered, the core interrupt service routine (ISR) checks to confirm if current frame processing is complete. It then calls a user programmable callback function, if any. In the callback function, the register settings for the next frame should be programmed, that is, what layer buffer address a layer should read next from. Finally, the interrupt service routine triggers the core to start processing the next frame.

  An application must perform the following tasks to configure the core for Interrupt mode.

- Register the core ISR routine `XVMix_InterruptHandler` with the system interrupt controller.
- Register the application callback function that should be called within the interrupt context. This can be done using the API `XVMix_SetCallback`.
- Enable the interrupts by calling provided API `XVMix_InterruptEnable`.

Usage

To better understand the driver usage, consider the following test case scenario. Suppose the core in the design was configured with few memory layers and each layer has certain optional features, like alpha blending and/or scaling enabled, and that the logo layer is enabled with the optional color key feature. Because memory layers are being used, there are sources in the design that generates frame data for each of the Video Mixer memory layers. The application should allocate required frame buffer space per layer in memory. These addresses should be updated during the interrupt.

To integrate and use the Video Mixer driver in the application, the following steps should be followed:

1. Include the driver header file `xv_mix_l2.h` that contains the mixer instance object definition.
2. Declare an instance of the Video Mixer type: `XV_Mix_l2 Mixer;`
3. Initialize the Video Mixer instance at power on:

   ```
   int XVMix_Initialize(XV_Mix_l2 *InstancePtr, u16 DeviceId);
   ```

   This function accesses the hardware configuration and initializes the core to the power on default state.

   - Set Master layer to 1080p.
   - Set Background color to blue.
   - Enable the master layer.
   - Set the operating mode to **Auto Restart**.
4. If the core is operating in interrupt mode, the application needs to perform the tasks mentioned, that is, register the ISR with the system interrupt controller and set the application callback function. This function is called by the Video Mixer driver when the frame done IRQ is triggered.
5. If applicable, write the application level callback function. An example action to be performed here would be to update layer buffer addresses from where to read the next frame data for each layer. This allows the application to render the frame updates in memory, on screen.

6. Write a function to configure the core. The following is a sample event sequence that might be performed here:

   a. Set the master layer video stream properties:

```
void XVMix_SetVidStream(XV_Mix_l2 *InstancePtr,
                        const XVidC_VideoStream *StrmIn);
```

   b. For each memory layer, set the frame buffer base address:

```
int XVMix_SetLayerBufferAddr(XV_Mix_l2 *InstancePtr,
                             XVMix_LayerId LayerId,
                             u32 Addr);
```

   c. If logo layer is enabled, load the logo data:

```
int XVMix_LoadLogo(XV_Mix_l2 *InstancePtr,
                   XVidC_VideoWindow *Win,
                   u8 *RBuffer,
                   u8 *GBuffer,
                   u8 *BBuffer);
```

   d. If logo color key feature is enabled, set the default color key data:

```
int XVMix_SetLogoColorKey(XV_Mix_l2 *InstancePtr,
                          XVMix_LogoColorKey ColorKeyData);
```

   e. For each layer, set the window properties:

```
int XVMix_SetLayerWindow(XV_Mix_l2 *InstancePtr,
                         XVMix_LayerId LayerId,
                         XVidC_VideoWindow *Win,
                         u32 StrideInBytes);
```

   f. Enable the interrupts (if operating in interrupt mode):

```
void XVMix_InterruptEnable(XV_Mix_l2 *InstancePtr);
```

   g. Enable the master layer (and additional memory layers if needed):

```
int XVMix_LayerEnable(XV_Mix_l2 *InstancePtr, XVMix_LayerId LayerId);
```

   h. Finally, start the core:

```
void XVMix_Start(XV_Mix_l2 *InstancePtr);
```

7. If optional features like alpha blending or scaling are enabled for a given layer, then these can be updated using the provided APIs.

```
int XVMix_SetLayerAlpha(XV_Mix_l2 *InstancePtr,
                        XVMix_LayerId LayerId,
                        u16 Alpha);
int XVMix_SetLayerScaleFactor(XV_Mix_l2 *InstancePtr,
                              XVMix_LayerId LayerId,
                              XVMix_Scalefactor Scale);
```

You are encouraged to experiment with other APIs that allow the layer window to be resized or moved on the screen or change logo color key information, if applicable. Each API provides a return value indicating if the desired action was successful or not.

Also, to help debug the Video Mixer, two Debug APIs are available that provide the state of the core:

```
void XVMix_DbgReportStatus(XV_Mix_l2 *InstancePtr);
void XVMix_DbgLayerInfo(XV_Mix_l2 *InstancePtr, XVMix_LayerId LayerId);
```

Note:

a. Resolution changes are not possible on the fly. If either the master input stream resolution changes during operation or the output resolution needs to be changed, then the application must reset the Video Mixer, that is, toggle `ap_rst_n`, and reconfigure the core for the new resolution. After reset, all registers are cleared to 0.

b. Certain actions cannot be performed when the core is in middle of processing a frame. For example, if a window has to be resized or the scaling factor has to be changed for a layer, then this layer should be disabled first. Next, apply the new settings and lastly re-enable the layer. Alternatively, in interrupt mode these operations can be done in the user callback function registered with the mixer interrupt service routine.

c. When resizing, moving, or scaling a layer window, the driver checks to ensure that the window properties provided does not cause the new window to go out of frame boundary. If any of these actions cause such a condition, then the action cannot applied and the API returns with an error code. The application code should check the return status of all APIs to make sure the required action was completed successfully and if not take corrective action.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the Video Mixer, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the Video Mixer. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the Video Mixer

AR: 66753

Technical Support

Xilinx provides technical support at the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

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

Vivado Design Suite Debug Feature

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

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

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 7].

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the debug feature for debugging the specific problems.

General Checks

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

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

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. *Vivado Design Suite: AXI Reference Guide* (UG1037)
2. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
3. *Vivado Design Suite User Guide: Designing with IP* (UG896)
4. *Vivado Design Suite User Guide: Getting Started* (UG910)
5. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
6. *ISE to Vivado Design Suite Migration Guide* (UG911)
7. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
8. *Vivado Design Suite User Guide: Implementation* (UG904)
9. *Video Processing Subsystem Reference Design Application Note* (XAPP1291)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/05/2017	1.0	• Added BGRA8, Y_UV10, Y_UV10_420, Y8, and Y10 to Memory Mapped AXI4 Interface.
10/05/2016	1.0	• Added YUV 4:2:0. • Updated Features in IP Facts. • Updated SDK directory link in IP Facts table. • Updated Feature Summary section. • Added RGBA8 to Y_UV8_420 sections. • Updated description for Layer Buffer (0x0048+i*8) Register section. • Added 0x4 0000 Logo Alpha Buffer table. • Updated description in Logo Red Buffer (0x1 0000) Register. • Updated description in General Design Guidelines section. • Updated Alpha Blending section. • Updated description to Layer Settings and Logo Layer Settings in Design Flow Steps chapter. • Added Enable Logo per Pixel Alpha in Vivado IDE Parameter to User Parameter Relationship table. • Updated Prerequisites section.
04/06/2016	1.0	Initial Xilinx release.

Please Read: Important Legal Notices