

Video Frame Buffer Read v2.0 and Video Frame Buffer Write v2.0

LogiCORE IP Product Guide

Vivado Design Suite

PG278 April 4, 2018

Table of Contents

IP Facts

Chapter 1: Overview

Feature Summary	5
Applications	5
Licensing and Ordering Information	6

Chapter 2: Product Specification

Standards	7
Performance	7
Resource Utilization	7
Port Descriptions	8
Register Space	18

Chapter 3: Designing with the Core

General Design Guidelines	23
Clock, Enable, and Reset Considerations	24

Chapter 4: Design Flow Steps

Customizing and Generating the Core	26
Constraining the Core	32
Simulation	33
Synthesis and Implementation	33

Chapter 5: Example Design

Frame Buffer Read Example Design	35
Frame Buffer Write Example Design	37

Chapter 6: Test Bench

Appendix A: Verification, Compliance, and Interoperability

Simulation	42
Hardware Testing	42
Interoperability	43

Appendix B: Migrating and Upgrading

Upgrading in the Vivado Design Suite 44

Appendix C: Application Software Development

Building the Board Support Package 46
Prerequisites 46
Modes of Operation..... 47
Usage 48

Appendix D: Debugging

Finding Help on Xilinx.com 50
Debug Tools 51
Hardware Debug 52

Appendix E: Additional Resources and Legal Notices

Xilinx Resources 53
Documentation Navigator and Design Hubs 53
References 53
Revision History 54
Please Read: Important Legal Notices 54

Introduction

The Xilinx® LogiCORE™ IP Video Frame Buffer Read and Video Frame Buffer Write cores provide high-bandwidth direct memory access between memory and AXI4-Stream video type target peripherals, which support the AXI4-Stream Video protocol.

Features

- AXI4 Compliant
- Streaming Video Format support for: RGB, RGBA, YUV 4:4:4, YUVA 4:4:4, YUV 4:2:2, YUV 4:2:0
- Memory Video Format support for: RGBX8, BGRX8, YUVX8, YUYV8, UYVY8, RGBA8, BGRA8, YUVA8, RGBX10, YUVX10, Y_UV8, Y_UV8_420, RGB8, BGR8, YUV8, Y_UV10, Y_UV10_420, Y8, Y10
- Provides programmable memory video format
- Supports progressive and interlaced video
- Supports 8 and 10-bits per color component on stream interface and memory interface
- Supports spatial resolutions from 64 x 64 up to 8192 x 4320
- Supports 4K60 in all supported device families⁽¹⁾

1. Performance on low power devices may be lower.

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 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 Linux DMA Controller
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](#).

Overview

Many video applications require frame buffers to handle frame rate changes or changes to the image dimensions (scaling or cropping). The Video Frame Buffer Read and Video Frame Buffer Write IP cores are designed to allow for efficient high-bandwidth access between the AXI4-Stream video interface and the AXI4 interface.

Feature Summary

The Video Frame Buffer Read and Video Frame Buffer Write are independent IPs which support reading and writing a variety of video formats (RGB, YUV 4:4:4, YUV 4:2:2, YUV 4:2:0, and Luma only). The data is packed/unpacked based on the video format. Planar and semi-planar memory formats are available for YUV 4:2:2 and YUV 4:2:0. The memory video format, stride, and frame buffer address are run time programmable.

Applications

Applications range from video systems for broadcast, professional Audio/Video, consumer medical, surveillance and industrial applications which include the following functionality:

- Video switchers, format converters
- Video CODECs
- Video surveillance systems
- Machine vision systems
- Video conferencing end unit
- Set-top box

Licensing and Ordering Information

This Xilinx LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the [Xilinx End User License](#). Information about this and other Xilinx LogiCORE IP modules is available at the [Xilinx Intellectual Property](#) page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your [local Xilinx sales representative](#).

For more information, visit the Video Frame Buffer [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 Frame Buffer Read and Video Frame Buffer Write cores are 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 Frame Buffer Read and Video Frame Buffer Write cores.

Maximum Frequencies

The following are typical clock frequencies for the target devices:

- UltraScale+ devices with -1 speed grade or higher: 300 MHz
- 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.

Resource Utilization

For full details about performance and resource utilization, visit the [Video Frame Buffer Read Performance and Resource Utilization web page](#) and the [Video Frame Buffer Write Performance and Resource Utilization web page](#).

Port Descriptions

The Video Frame Buffer Read and Video Frame Buffer Write cores use industry standard control and data interfaces to connect to other system components. The following sections describe the various interfaces available with the cores. [Figure 2-1](#) illustrates the Video Frame Buffer Read diagram. Each IP has three AXI interfaces:

- AXI-Lite control interface (s_axi_CTRL)
- AXI4-Stream streaming video output (m_axis_video) or input (s_axis_video)
- Memory mapped AXI4 interface (m_axi_mm_video)

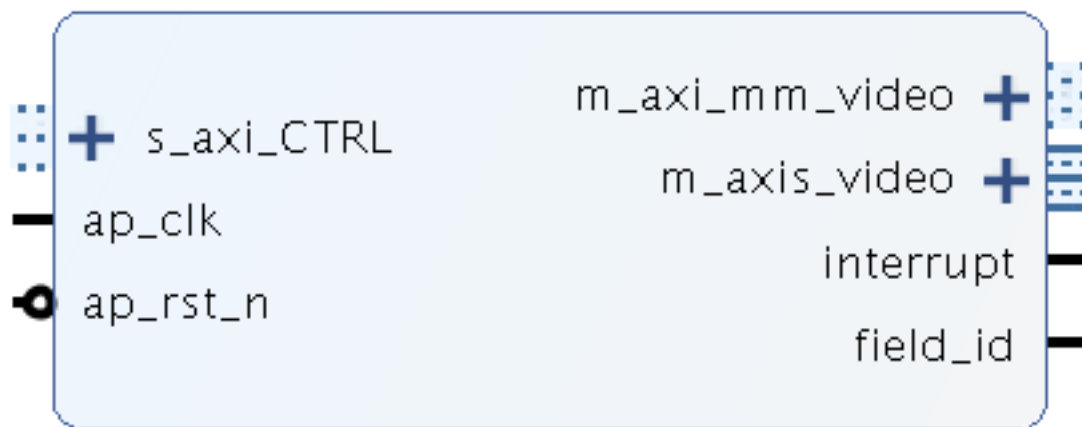


Figure 2-1: Video Frame Buffer Read I/O Diagram

Figure 2-2 illustrates the Video Frame Buffer Write diagrams.



Figure 2-2: Video Frame Buffer Write I/O Diagram

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
<code>ap_clk</code>	I	1	Video core clock
<code>ap_rst_n</code>	I	1	Video core active-Low clock enable
<code>interrupt</code>	O	1	Interrupt Request Pin
<code>field_id</code>	O (Frame Buffer Read)	1	Field polarity (only available when interlaced support is selected)
	I (Frame Buffer Write)		

The `ap_clk` and `ap_rst_n` signals are shared between the core, the AXI4-Stream, memory mapped AXI4 data interface, 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 allow for interrupt aggregation to the system processor.

field_id

The `field_id` signal indicates the polarity of the field when the video is interlaced. This signal is only used with interlaced data. This signal is ignored for progressive video inputs. The `field_id` signal changes with the rising edge of Start of Frame/Field (`TUSER`) of the AXI4-Stream interface.

AXI4-Stream Video Interface

The Video Frame Buffer Read and Video Frame Buffer Write cores have either an AXI4-Stream video input or output interface named `s_axis_video` or `m_axis_video`, respectively. All video streaming interfaces follow the interface specification as defined in the *Video IP* section 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 or 10 bits per component.

Table 2-2 through Table 2-6 explain the pixel mapping of an AXI4-Stream interface with two pixels per clock and 10 bits per component configuration for all supported color formats. Given that the Video Frame Buffer Read and Video Frame Buffer Write always require a hardware configuration of three component video, the AXI4-Stream Subset Converter is needed to hook up with other IPs of two or one component video interface in YUV 4:2:2, YUV 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: Dual Pixels per Clock, 10 Bits per Component Mapping for YUV 4:2:0, for Even Lines

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 2-6: Dual Pixels per Clock, 10 Bits per Component Mapping for YUV 4:2:0, for Odd Lines

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

This IP always generates three video components even if the video format is set to be YUV 4:2:0 or YUV 4:2:2 at run time. The unused components can be set to zero.

Table 2-7 shows the interface signals for input and output AXI4-Stream video streaming interfaces.

Table 2-7: AXI4-Stream Interface Signals

Signal Name	I/O	Width	Description
s_axis_tdata	I	$\text{floor}(\frac{\text{number_of_components} \times \text{bits_per_component} \times \text{pixels_per_clock} + 7}{8}) \times 8$	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	$(\text{s_axis_video_tdata width}) / 8$	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	$(\text{s_axis_video_tdata width}) / 8$	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	$\text{floor}(\frac{\text{number_of_components} \times \text{bits_per_component} \times \text{pixels_per_clock} + 7}{8}) \times 8$	Output Data
m_axis_tdest	O	1	Output Data Routing Identifier
m_axis_tid	O	1	Output Data Stream Identifier
m_axis_tkeep	O	$(\text{m_axis_video_tdata width}) / 8$	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

Table 2-7: AXI4-Stream Interface Signals (Cont'd)

Signal Name	I/O	Width	Description
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

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]. The Video Frame Buffer Read and Video Frame Buffer Write support the pixel formats in memory described in Table 2-8.

Table 2-8: Pixel Formats

Video Format	ID	Description	Bits per Component	Bytes per Pixel
RGBX8	10	packed RGB	8	4 bytes per pixel
BGRX8	27	packed BGR	8	4 bytes per pixel
YUVX8	11	packed YUV 4:4:4	8	4 bytes per pixel
YUYV8	12	packed YUV 4:2:2	8	2 bytes per pixel
UYVY8	28	packed YUV 4:2:2	8	2 bytes per pixel
RGBA8 ⁽¹⁾	13	packed RGB with alpha	8	4 bytes per pixel
BGRA8 ⁽¹⁾	26	packed BGR with alpha	8	4 bytes per pixel
YUVA8 ⁽¹⁾	14	packed YUV 4:4:4 with alpha	8	4 bytes per pixel
RGBX10	15	packed RGB	10	4 bytes per pixel
YUVX10	16	packed YUV 4:4:4	10	4 bytes per pixel
Y_UV8	18	semi-planar YUV 4:2:2	8	1 byte per pixel per plane
Y_UV8_420	19	semi-planar YUV 4:2:0	8	1 byte per pixel per plane
RGB8	20	packed RGB	8	3 bytes per pixel
BGR8	29	packed BGR	8	3 bytes per pixel
YUV8	21	packed YUV 4:4:4	8	3 bytes per pixel
Y_UV10	22	semi-planar YUV 4:2:2	10	4 bytes per 3 pixels per plane
Y_UV10_420	23	semi-planar YUV 4:2:0	10	4 bytes per 3 pixels per plane
Y8	24	packed luma only	8	1 byte per pixel

Table 2-8: Pixel Formats (Cont'd)

Video Format	ID	Description	Bits per Component	Bytes per Pixel
Y10	25	packed luma only	10	4 bytes per 3 pixels

Notes:

1. Per pixel alpha memory formats are only available with Video Frame Buffer Read IP.

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

Note: RGB formats are stored in memory in RGB order which is different than the pixel mapping in *AXI4-Stream Video IP and System Design* (UG1037) [Ref 1].

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

BGRX8

Packed BGR, 8 bits per component. Every BGR 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	R	G	B

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 YUV 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

UYVY8

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

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

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

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. See *Stride* in [Prerequisites in Appendix C](#) for more information on stride.

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. See *Stride* in [Prerequisites in Appendix C](#) for more information on stride.

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

BGR8

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

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

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. See *Stride* in [Prerequisites in Appendix C](#) for more information on stride.

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. See *Stride* in [Prerequisites in Appendix C](#) for more information on stride.

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 Frame Buffer Read and Video Frame Buffer Write cores can be controlled through the AXI4-Lite interface by using functions provided by the driver in the Software Development Kit (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 2-9](#) shows the AXI4-Lite control interface signals. This interface runs at the `ap_clk` clock.

Table 2-9: 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

Table 2-9: AXI4-Lite Control Interface Signals (Cont'd)

Signal Name	I/O	Width	Description
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
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 Frame Buffer Read and Video Frame Buffer Write cores have specific registers which allow you to dynamically control the operation of the cores. All registers have an initial value of 0.

Top-Level Registers

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

Table 2-10: Top-Level Registers

Address (hex) BASEADDR+	Register Name	Access Type	Register Description
0x0000	Control	R/W	Bit [0] = ap_start (R/W/COH) ⁽¹⁾ Bit [1] = ap_done (R/COR) Bit [2] = ap_idle (R) Bit [3] = ap_ready (R) Bit [5] = Flush pending AXI transactions Bit [6] = Flush done Bit [7] = auto_restart (R/W) Others = reserved
0x0004	Global Interrupt Enable	R/W	Bit [0] = Global interrupt enable Others = Reserved
0x0008	IP Interrupt Enable	R/W	Bit [0] = ap_done Bit [1] = ap_ready Others: reserved
0x000c	IP Interrupt Status	R/TOW ⁽¹⁾	Bit [0] = ap_done Bit [1] = ap_ready Others: reserved
0x0010	Width	R/W	Active width of stream on s_axis_video or m_axis_video
0x0018	Height	R/W	Active height of stream on s_axis_video or m_axis_video
0x0020	Stride	R/W	Active stride (in bytes)
0x0028	Memory Video Format	R/W	Active video format of data in memory
0x0030	Plane 1 Buffer	R/W	Start address of plane 1 of frame buffer
0x003C	Plane 2 Buffer	R/W	Start address of plane 2 of frame buffer. Only valid when a semi-planar video format is selected.
0x0048	Field ID	R/W	Field polarity. Only valid for interlaced video.

Notes:

1. COH = Clear on Handshake, COR = Clear on Read, TOW = Toggle on Write

Control (0x0000) Register

This register controls the operation of the core. 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. Bit[5] is for flushing pending AXI transactions. This bit should be set and held (until reset) by software to flush pending transactions. When this is set, the hardware is expecting a hard reset. Bit[6] is the flush status bit and is asserted when the flush is done.

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 Scanline. 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 (or per field for interlaced video). 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. For interlaced data, the height for each field must be the same.

Stride (0x0020) Register

The stride determines the number of bytes between rows of pixels in memory. When a video frame is stored in memory, the memory buffer may contain extra padding bytes after each row of pixels. The padding bytes only affect how the image is stored in memory, but do not affect how the image is displayed.

Padding bytes is necessary to ensure 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, the stride value must be a multiple of the memory mapped AXI4 data size. The data size of the memory mapped AXI4 interface is $64 \times \text{Samples per Clock}$ bits, for example 64, 128, or 256 bits for 1, 2, and 4 samples per clock, respectively.

Memory Video Format (0x0028) Register

This register specifies the video format that the core uses for memory. Supported memory video formats are those selected in the Vivado IDE (See Table 2-8). The memory video format should match the AXI4-Stream interface data format as show in Table 2-11. Memory formats using 8 bits per component can be scaled up and then read out to a 10-bit per component video stream. (Scaling is performed by shifting bits. For example, 255 in 8 bits becomes 1020 in 10 bits.) Similarly, a 10-bit video stream can be scaled down and written to 8-bit memory video formats. The reverse is not true. 8-bit video stream data cannot be written to or read from 10-bit memory video formats.

Table 2-11: Compatible Memory and Streaming Video Formats

Memory Video Format	Compatible AXI4-Stream Video Format
RGBX8	RGB
RGBX10	
RGB8	
BGR8	
BGRX8	
YUVX8	YUV 4:4:4
YUVX10	
YUV8	
Y8	
Y10	
YUYV8	YUV 4:2:2
Y_UV8	
Y_UV10	
UYVY8	
Y_UV8_420	YUV 4:2:0
Y_UV10_420	
RGBA8	RGB with per pixel alpha
BGRA8	
YUVA8	YUV 4:4:4 with per pixel alpha

Plane1 Buffer (0x0030) and Plane 2 Buffer (0x003C) Registers

The Plane 1 Buffer register specifies the frame buffer address of plane 1. Note that for the semi-planar formats (Y_UV8, Y_UV8_20, Y_UV10, Y_UV10_420), the chroma buffer is specified by the Plane 2 Buffer register. The address needs to be aligned to the data size of the memory mapped AXI4 interface. The data size of the memory mapped AXI4 interface is 64***Samples per Clock** bits, for example 64, 128, or 256 bits for 1, 2, and 4 samples per clock, respectively.

Field ID (0x0048) Register

Field ID is used to identify if the field being read/written is even or odd. This register is only available with interlaced video. The register is R/W for the Frame Buffer Read, and Read only for the Frame Buffer Write.

Designing with the Core

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

General Design Guidelines

This section compares the differences between Video Frame Buffer Read and Video Frame Buffer Write IPs to the AXI Video Direct Memory Access IP core.

The AXI Video Direct Memory Access (VDMA) does not perform pixel packing or unpacking of data it reads and writes to and from memory. Data is written to memory as it is formatted on the `AXIS_VIDEO_TDATA` bus. This has the following consequences:

- 8-bit RGB is written to memory as GBR.
- YUV 4:2:2 over a three component streaming interface is written to memory as three components although the third component does not contain any valid data.
- YUV 4:2:0 is the same as YUV 4:2:2. This means 4:2:0 takes twice the bandwidth compared to separating out luma and chroma planes.
- 10-bit formats are written differently to memory depending on pixels per clock. For example, with one pixel per clock, RGBX is written as:

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

Whereas with two pixels per clock, RGBX is written as:

63:60	59:50	49:40	39:30	29:20	19:10	9:0
X	R	B	G	R	B	G

Video Frame Buffer Read and Video Frame Buffer Write IPs have some differences/limitations when compared to AXI VDMA. The following list highlights some key differences between Video Frame Buffer IP and AXI VDMA.

- Video Frame Buffer Read and Video Frame Buffer Write cores support single channel read or single channel write, but never combined. The AXI VDMA supports combined read and write channels in one IP.
- Because of the previous point, Video Frame Buffer Read and Video Frame Buffer Write cores do not use Genlock Synchronization (prevents reading and writing from and to one buffer). You should implemented this in software.
- Video Frame Buffer Read and Video Frame Buffer Write cores currently only support 8 and 10 bits per component.
- Video Frame Buffer Read and Video Frame Buffer Write cores do not allow for data realignment to the byte (8 bits) level. For example, the addresses must be aligned to multiples of memory mapped data width bytes.
- Video Frame Buffer Read and Video Frame Buffer Write cores do not allow for explicit control of the size of the memory bus. If data width conversion is required, use an AXI interconnect IP.



RECOMMENDED: *Xilinx recommends using the Video Frame Buffer Read and Video Frame Buffer Write for reading from or writing to memory when using Video IP such as the Video Processing Subsystem. For Zynq UltraScale+ MPSoC devices, the Video Codec Unit can properly encode and decode the video data in memory.*

Clock, Enable, and Reset Considerations

Clocking

The Video Frame Buffer Read and Video Frame Buffer Write IPs have a single clock domain. All interfaces (master and slave AXI4-Stream video interfaces, the AXI4-Lite interface, and the memory mapped AXI4 interfaces) use the `ap_clk` pin as their clock source.

Resets

The Video Frame Buffer Read and Video Frame Buffer Write IPs have only a hardware reset option, `ap_rst_n` pin. No software reset option is available. The external reset pulse must 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

To operate properly, the Video Frame Buffer read and write IPs must be configured for the actual input and output image frame size. To gather the frame size information from the image video stream, the IP can be connected to the Video In to AXI4-Stream input and the Video Timing Controller core. 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 Frame Buffer Read and Video Frame Buffer Write, with the appropriate image dimensions.

Ensure that there is sufficient bandwidth available to the Video Frame Buffer Read and Video Frame Buffer Write. The bandwidth required (in MB/s) can be calculated with the following equation:

$$\text{Bandwidth (MB/s)} = \text{fps} \times \text{height} \times \text{stride} \quad \text{Equation 3-1}$$

where *fps* is the number of frames per second the Video Frame Buffer Read and Video Frame Buffer Write is operating at; *height* is the height in lines; *stride* is the stride value in bytes. See *Stride* in [Prerequisites in Appendix C](#) for more information on stride.

Programming Sequence

The IP's `Buffer` parameters can be changed dynamically and the change is picked up immediately. If the `Width`, `Height`, `Stride`, or `Video Format` must be changed or the entire system 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.

Pending AXI transactions can be flushed before resetting the IP. Assert and hold Bit[5] of the Control register. Bit[6] asserts when the flush is complete and then the IP can be reset.

For interlaced video, with the `ap_ready` interrupt, the current Field ID can be read for the Video Frame Buffer Write IP, and the next Field ID can be programmed for the Frame Buffer Read IP.

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: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). The layout depicted here might vary from the current version.

Interface

This section provides a quick reference to parameters that can be configured using the Vivado Design Suite at generation time.

[Figure 4-1](#) and [Figure 4-2](#) show the Video Frame Buffer Read and Video Frame Buffer Write Vivado IDE main configuration screens, respectively.

Video Frame Buffer Read (2.0)

Documentation IP Location Switch to Defaults

Show disabled ports

Component Name

Samples per Clock

Maximum Number of Columns [64 - 8192]

Maximum Number of Rows [64 - 4320]

Maximum Data Width

Address Width

Video Formats with Alpha

Interlaced Support

Video Formats with per pixel Alpha

RGBA8 BGRA8

YUVA8

8 Bit Video Formats

RGBX8 RGB8 BGRX8 BGR8

YUVX8 YUV8

YUYV8 UYVY8 Y_UV8

Y_UV8_420

Y8

10 Bit Video Formats

RGBX10

YUVX10

Y_UV10

Figure 4-1: Video Frame Buffer Read Customize IP

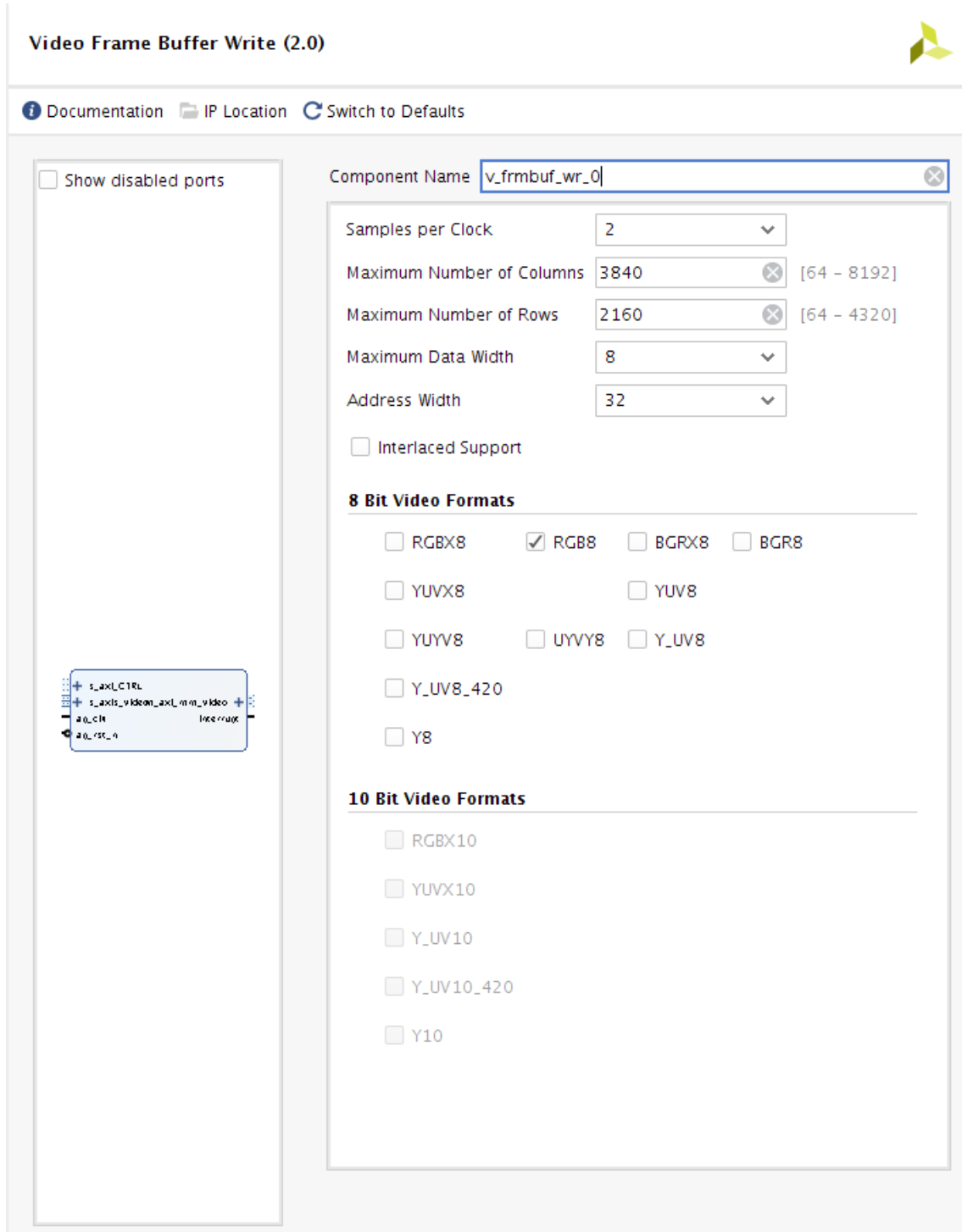


Figure 4-2: Video Frame Buffer Write Customize IP

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 "_".
- **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.
- **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.
- **Maximum Data Width**: Specifies the bit width of input and output samples on all the streaming interfaces. Permitted values are 8 and 10 bits. This parameter should match the Video Component Width of the video IP core connected to the AXI-Stream video interface.
- **Address Width**: Specifies the address width of the AXI master interfaces for the memory interface, either 32 or 64 bits.
- **Interlaced Support**: Select for support of interlaced video (in addition to progressive video).
- **Video Formats**: Select the memory video formats to be supported. The video format can be programmed through the AXI4-Lite control interface. Details on each video format can be found in [Memory Mapped AXI4 Interface in Chapter 2](#).

Three video formats support per pixel alpha: RGBA8, BGRA8, and YUVA8. These video formats are only available with the Video Frame Buffer Read IP.

Both packed and semi-planar formats are offered. If only packed formats are selected, the IP will be smaller in size because there is only one plane. The following formats are packed and only need one plane:

- RGBX8
- BGRX8
- YUVX8
- YUYV8

- UYVY8
- RGBA8
- BGRA8
- YUVA8
- RGBX10
- YUVX10
- RGB8
- BGR8
- YUV8
- Y8
- Y10

The following formats are semi-planar and require two planes:

- Y_UV8
- Y_UV8_420
- Y_UV10
- Y_UV10_410

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
Number of Video Components	NUM_VIDEO_COMPONENTS	3
Samples per Clock	SAMPLES_PER_CLOCK	2
Maximum Number of Columns	MAX_COLS	3840
Maximum Number of Rows	MAX_ROWS	2160
Maximum Data Width	MAX_DATA_WIDTH	8
AXIMM Data Width	AXIMM_DATA_WIDTH	128
AXIMM Address Width	AXIMM_ADDR_WIDTH	32
Number Read/Write Outstanding	AXIMM_NUM_OUTSTANDING	4
Transaction Burst Length	AXIMM_BURST_LENGTH	16
Video Formats with per pixel Alpha	HAS_ALPHA	0
Interlaced Support	HAS_INTERLACED	0

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

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
RGBX8	HAS_RGBX8	0
BGRX8	HAS_BGRX8	0
YUVX8	HAS_YUVX8	0
YUYV8	HAS_YUYV8	0
UYVY8	HAS_UYVY8	0
RGBA8	HAS_RGBA8	0
BGRA8	HAS_BGRA8	0
YUVA8	HAS_YUVA8	0
RGBX10	HAS_RGBX10	0
YUVX10	HAS_YUVX10	0
Y_UV8	HAS_Y_UV8	0
Y_UV8_420	HAS_Y_UV8_420	0
RGB8	HAS_RGB8	1
BGR8	HAS_BGR8	0
YUV8	HAS_YUV8	0
Y_UV10	HAS_Y_UV10	0
Y_UV10_420	HAS_Y_UV10_420	0
Y8	HAS_Y8	0
Y10	HAS_Y10	0
Maximum Number of Pixel Planes	MAX_NR_PLANES	1

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: one containing the Video Frame Buffer Read and another one containing both the Video Frame Buffer Read and the Video Frame Buffer Write cores. Important system-level aspects when designing with the cores are highlighted in the example designs, including:

- Typical usage of the Video Frame Buffer Read and Video Frame Buffer Write in conjunction with other cores.
- Run-time configuration of the Video Frame Buffer Read and Video Frame Buffer Write by programming registers on-the-fly.

The supported platforms are listed in [Table 5-1](#).

Table 5-1: Supported Platforms

Development Boards	Additional Hardware	Processor
KC705	N/A	MicroBlaze™
ZCU102	N/A	R5
ZCU104	N/A	R5
ZCU106	N/A	R5

To open the example project, perform the following:

1. Select the **Video Frame Buffer Read IP** or **Video Frame Buffer Write 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 Frame Buffer Read** or **Video Frame Buffer Write** 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.

Frame Buffer Read Example Design

The example design delivered with the Frame Buffer Read IP contains the following video IP cores: Video Frame Buffer Read, Video Timing Controller, and AXI4-Stream to Video Output bridge. The design also contains a processor (to enable register programming) connected to an AXI Interconnect. A Memory Interface Generator (MIG) is used for DDR memory access.

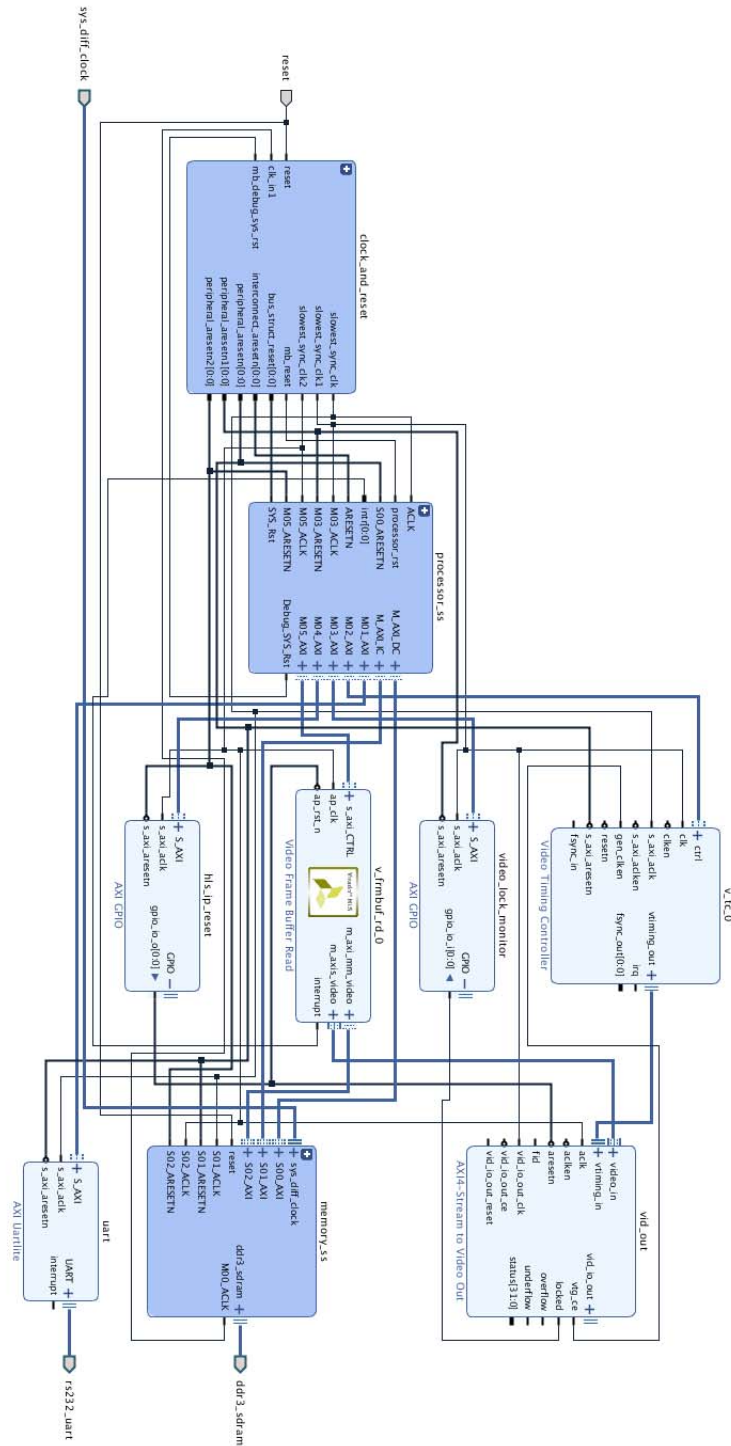


Figure 5-1: Frame Buffer Read Example Design

The processor acts as the system's AXI master and drives the Video Frame Buffer Read and Video Timing Controller cores. It configures height, width, and other registers of the Video Frame Buffer Read, then configures the timing parameters of the Video Timing Controller core.

The Video Frame Buffer Read core generates video stream pixels at a clock rate of `ap_clk`. The core reads frames from a memory mapped AXI4 memory interface and sends the data 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 example design checks the LOCKED output from the AXI4-Stream to Video Out core. A locked port indicates that the output timing is locked to the output video. The example design indicates that the test completed successfully if video lock is successfully detected. The locked port of AXI4-Stream to Video Out is connected to AXI GPIO core and the processor polls the corresponding register for a sign that the test passed.

Frame Buffer Write Example Design

The Video Frame Buffer Write core has a separate example design. The difference between the example design delivered with Video Frame Buffer Write and the example design delivered with Video Frame Buffer Read is that the Video Frame Buffer Write design also contains the Video Frame Buffer Write and Video Input to AXI4-Stream bridge. The example design checks the overflow port from the Video Input to AXI4-Stream for data being dropped and not consumed in a timely fashion by the Video Frame Buffer Write core. The overflow port of the Video Input to AXI4-Stream core is connected to the AXI GPIO for the processor to monitor.

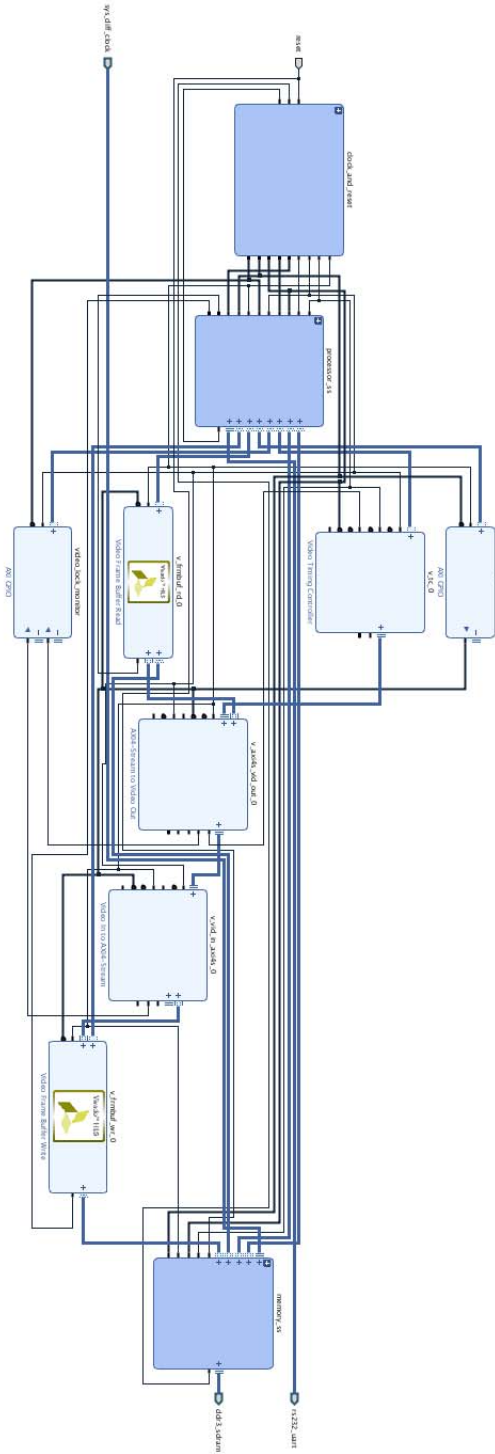


Figure 5-2: Frame Buffer Write Example Design

Running the Example Design

The synthesizable example design requires both Vivado and Xilinx SDK tools. To run the example design, perform the following:

1. Run synthesis, implementation and bitstream generation in Vivado.
2. After completing step 1, select **File -> Export -> Export Hardware**. In the window, select **Include bitstream**, select an export directory and click **OK**.

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

```
(/data/embeddedsw/XilinxProcessorIPLib/drivers/v_frmbuf_rd_v3_0/examples/  
(/data/embeddedsw/XilinxProcessorIPLib/drivers/v_frmbuf_wr_v3_0/examples/
```

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

The `v_frmbuf_rd_example.tcl` and `v_frmbuf_wr_example.tcl` files automate 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 in xsct:

```
%>source vfrmbuf_xx_example.tcl
```

where `xx` is `rd` for Video Frame Buffer Read and `wr` for Video Frame Buffer Write.

5. Execute the script:

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

The Tcl script performs the following:

- Create workspace
- Create hardware project
- Create Board Support Package
- Create Application Project
- Build BSP and Application Project

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

```
bit file -> vfrmbuf_xx_example.sdk/vfrmbuf_xx_example_hw_platform folder
elf file -> vfrmbuf_xx_example.sdk/vfrmbuf_xx_example_design/{Debug/Release} folder
```

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



IMPORTANT: *To do so, ensure 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 `vfrmbuf_xx_example.sdk` folder in prompted window. The SDK project opens automatically. (If a welcome page displays, close it.)
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 the Tcl script, then click **Program**.
Note: The DONE LED on the board turns green if the programming is successful.
5. A terminal program (such as HyperTerminal or PuTTY) is needed for UART communication. Open the program, choose appropriate port, set baud rate to 115200 and establish Serial port connection.
6. Select and right-click the application `vfrmbuf_xx_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 the `readme.txt` in the examples folder. Video inputs tested are 4Kp60, 4Kp30, 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 Frame Buffer Read and Video Frame Buffer Write 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 Frame Buffer Read and Video Frame Buffer Write with AXI4-Stream and memory mapped AXI4 interface
 - Testing of various frame sizes
 - Varying parameter settings
-

Hardware Testing

The Video Frame Buffer Read and Video Frame Buffer Write cores have been validated at Xilinx® to represent many different parameterizations. A test design was developed for the core that incorporated a processor, AXI4-Lite interconnect, and various other peripherals. The processor was responsible for:

- Programming 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 provides information about any changes to the user logic or port designations between core versions.

Changes from v2.0 (Rev. 1) (10/04/2017) to v2.0 (Rev. 2) (04/04/2018)

Parameters Changes

Two new parameters were added:

- HAS_BGR8: The memory format BGR8 has been added.
- HAS_INTERLACED: Support for interlaced data in addition to progressive data has been added.

Port Changes

The port `field_id` is added when support for interlaced video is selected.

Other Changes

v2.0 (Rev. 2)

- In v2.0 (Rev. 2) (04/04/2018), there are APIs for both `ap_done` interrupts and `ap_ready` interrupts. The API function calls for the interrupt enable and disable have changed. An additional argument is needed to define a mask for which interrupt to enable/disable. The `SetCallback` function also has an additional argument to specify the interrupt source.
- In v2.0 (Rev. 2) (04/04/2018), the AXI MM interface always operates in conservative mode.

The HLS AXI master interface always checks the data buffer on W channel before it issues a write request. Only when the buffer has more than burst length data is AWWVALID asserted. This can help to solve deadlock in some systems, but, usually the write latency will increase.

- In v2.0 (Rev. 2) (04/04/2018), pending AXI transactions can be flushed before reset.

v2.0 (Rev. 1)

- In v2.0 (Rev. 1) (10/04/2017), only the ap_done interrupt was supported in the driver APIs.
- In v2.0 (Rev. 1) (10/04/2017) does not use conservative mode:

The HLS AXI master interface issued a write request on AW channel without checking the data buffer on W channel.

Changes from v1.0 to v2.0

Parameter Changes

New parameters: HAS_ALPHA, HAS_RGBA8, HAS_BGRA8, HAS_YUVA8. The Video Frame Buffer Read IP supports per pixel Alpha. Two new streaming video formats, RGB with Alpha and YUV 4:4:4 with Alpha, are now available for use with the memory formats RGBA8, BGRA8, and YUVA8.

New parameters: HAS_BGRX8, HAS_UYVY8. New memory video formats have been added to both the Read and Write IPs: BGRX8 and UYVY8.

New parameter: AXIMM_ADDR_WIDTH. Both 32-bit and 64-bit addressing are supported for the memory interfaces.

Port Changes

There are no port changes.

Other Changes

Two buffers are provided for semi-planar formats. In v1.0 of the IP, the chroma buffer address was automatically set based on frame size and stride. In v2.0 of the IP, the chroma buffer address must be specified via a register.

Application Software Development

The Video Frame Buffer Read and Video Frame Buffer Write cores are 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 users 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 Frame Buffer Read and Video Frame Buffer Write cores. Xilinx® recommends always using layer 2 APIs to interact with the core.

Building the Board Support Package

When the Board Support Package is built, the Video Frame Buffer Read and Video Frame Buffer Write drivers inherently pull in the required dependency, that is, the video common driver. This driver contains enumerations for video specific information such as color format, color depth, frame rate, etc. It also defines fundamental data types to represent the video stream and timing that are used in the Video Frame Buffer Read and Video Frame Buffer Write drivers and is common across all Xilinx video drivers.

During the build process, the Video Frame Buffer Read and Video Frame Buffer Write drivers extract the Video Frame Buffer Read and Video Frame Buffer Write hardware configuration settings from the provided hardware design file.

Prerequisites

There are certain requirements that must be fulfilled when programming the core.

- The core does not contain a data realignment engine and therefore the application software must align the memory address before writing to the register. The alignment requirement is specified below. This ensures the start address is aligned with the width of the memory interface.

*Start address should be aligned with: (8 * Pixels per Clock) Bytes*

- The **Stride** value (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 the width in pixels the following equation can be used:

$$\text{Stride in Bytes} \geq (\text{Width} \times \text{Bytes per Pixel}); \quad \text{Equation C-1}$$

The bytes per pixel value varies per video in memory format, and is described in [Memory Mapped AXI4 Interface in Chapter 2](#).

Note: Padding bytes is sometimes necessary (hence the \geq 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.

- The **Width** value 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 Frame Buffer Read and Video Frame Buffer Write support two modes of operation.

- Synchronous Mode

When an interrupt is triggered, the core interrupt service routine (ISR) checks to confirm if current frame processing is complete, and then calls any additional user programmable callback functions. In the callback function, the register settings for the next frame should be programmed, that is, what buffer address should read next from.

For example, upon finishing a frame, the Video Frame Buffer Write IP triggers an interrupt at `ap_done`. At this time, the callback function is run to program a new buffer address. After running the callback function, the core restarts, and the next frame is processed.

- Asynchronous Mode

In asynchronous mode, the register settings for the next frame are programmed at any time rather than waiting first for an interrupt.

For example, The Video Frame Buffer Write IP finishes processing a frame and issues an interrupt. At this time, the buffer address is passed to the Video Frame Buffer Read IP which is in the middle of processing its own frame. The Video Frame Buffer Read buffer address is programmed asynchronously. When the Video Frame Buffer Read is finished processing its current frame, the newly programmed buffer address takes effect and the next frame is processed by the Video Frame Buffer Read IP.

Usage

To integrate and use the Video Frame Buffer Read and Video Frame Buffer Write core drivers in the application, perform the following steps:

Note: The Video Frame Buffer Read is used as an example.

1. Include the driver header file `xv_frmbufrd_l2.h` that contains the frame buffer instance object definition.

2. Declare an instance of the frame buffer core type:

```
XV_FrmbufRd_l2 FrmbufRd;
```

3. Initialize the frame buffer instance at power on.

```
int XVFrmbufRd_Initialize(XV_FrmbufRd_l2 *InstancePtr, u16 DeviceId);
```

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

4. The application must register the ISR with the system interrupt controller and set the application call back function. This function will then be called by the Video Frame Buffer Read or Video Frame Buffer Write driver when the IRQ is triggered.

- Register the core ISR routine `XVFrmbufRd_InterruptHandler` or `XVFrmbufWr_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 `XVFrmbufRd_SetCallback` or `XVFrmbufWr_SetCallback`.

5. If applicable, write the application level call back function. An example action to be performed here would be to update the buffer address from where to read the next frame data. This will allow the application to render the frame updates in memory, on screen.

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

- a. Set the video properties:

```
void XVFrmbufRd_SetMemFormat(XV_FrmbufRd_l2 *InstancePtr,
                             u32 StrideInBytes,
                             XVidC_ColorFormat MemFmt,
                             const XVidC_VideoStream *StrmIn);
```

- b. Set the frame buffer base address

```
int XVFrmbufRd_SetBufferAddr(XV_FrmbufRd_l2 *InstancePtr,
                              UINTPTR Addr);
```

For semi-planar formats, set the buffer base address for the second plane:


```
int XVFrmbufRd_SetChromaBufferAddr(XV_FrmbufRd_12 *InstancePtr,  
                                   UINTPTR Addr);
```

c. Enable the interrupts:

```
void XVFrmbufRd_InterruptEnable(XV_FrmbufRd_12 *InstancePtr, u32 IrqMask);
```

d. Start the core:

```
void XVFrmbufRd_Start(XV_FrmbufRd_12 *InstancePtr);
```

Also, to help debug the Video Frame Buffer Read core a Debug API is available that provides the state of the core.

```
void XVFrmbufRd_DbgReportStatus(XV_FrmbufRd_12 *InstancePtr);
```

Note:

- a. Resolution changes are not possible on-the-fly. If either the master input stream or the output stream resolution must be changed during operation, the application must reset the Video Frame Buffer Read and Video Frame Buffer Write, that is, toggle `ap_rst_n`, and reconfigure the core for the new resolution. After reset, all registers are cleared to 0.
- b. When configuring the frame buffer cores, the driver checks to ensure that the frame size, stride, and memory video format are valid. If any of these configurations are invalid, then the action will not be applied and the API will return 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 Frame Buffer Read and Video Frame Buffer Write, 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 Frame Buffer Read and Video Frame Buffer Write. 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 Frame Buffer Read

AR: [68764](#)

Master Answer Record for the Video Frame Buffer Write

AR: [68765](#)

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 Frame Buffer Read and Video Frame Buffer Write 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)

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.
- If the streaming data coming out of the Video Frame Buffer Read core has the wrong colors, check the Memory Video Format register and ensure that the proper video format used in memory is programmed in the register. Also, each memory video format has a corresponding streaming video format. For example, the memory format RGB8 outputs RGB streaming data. Ensure that the system is expecting the proper streaming video format.
- If the data written to memory by the Video Frame Buffer Write seems to be in the wrong format, check the Memory Video Format register. Ensure that the proper video format used in memory is programmed in the register.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

Note: For more information on Documentation Navigator, see the [Documentation Navigator](#) page on the Xilinx website.

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](#))

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/04/2018	2.0	<ul style="list-style-type: none"> • Added support for interlaced video • Added BGR8 video format • Added support for ZCU102, ZCU104, and ZCU106 boards in the example design.
10/04/2017	2.0	Added new video formats, added second buffer for semi-planar formats, added support for 64-bit addressing on memory interfaces.
04/05/2017	1.0	Initial Xilinx release.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2017-2018 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.