Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

Video Processing Subsystem v1.0

Product Guide

Vivado Design Suite

PG231 November 18, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

Appendix C: Application Software Development

Appendix D: Additional Resources and Legal Notices

Introduction

The Video Processing Subsystem is a collection of video processing IP subcores, bundled together in hardware and software, abstracting the video processing pipe. It provides the end-user with an out of the box ready to use video processing core, without having to learn about the underlying complexities. The Video Processing Subsystem enables streamlined integration of various processing blocks including (but not limited to) scaling, deinterlacing, color space conversion and correction, chroma resampling, and frame rate conversion.

Features

- One, two or four pixel-wide AXI4-Stream video interface
- Video resolution support up to UHD at the 60 fps
- Run-time color space support for RGB, YUV 4:4:4, YUV 4:2:2, YUV 4:2:0
- 8, 10, 12, and 16 bits per component support
- Deinterlacing
- Scaling
- Color space conversion and correction
- Chroma resampling between YUV4:4:4, YUV 4:2:2, YUV 4:2:0
- Frame rate conversion using dropped/ repeated frames

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families, UltraScale™ Architecture, Zynq®-7000, 7 Series
Supported User Interfaces	AXI4-Lite, AXI4-Stream, AXI-MM
Resources	See Table 2-1 through Table 2-2
Provided with Core	
Design Files	Encrypted HLS C
Example Design	Verilog
Test Bench	Not Provided
Constraints File	XDC
Simulation Model	Source HDL
Supported S/W Driver(2)	Standalone
Tested Design Flows(3)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Not Provided.
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. Standalone driver details can be found in the SDK directory (<*install_directory*>/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from the Xilinx Wiki page.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

Introduction

The Video Processing Subsystem enables streamlined integration of various processing blocks including (but not limited to) scaling, deinterlacing, color space conversion and correction, chroma resampling, and frame rate conversion.

Feature Summary

The Video Processing Subsystem has the following features:

- One, two, or four pixel-wide video interface
- Run-time color space support for RGB, YUV 4:4:4, YUV 4:2:2, YUV 4:2:0
- 8, 10, 12, and 16 bits per component support
- Deinterlacing
- Scaling
- Color space conversion and correction
- Chroma resampling between YUV4:4:4, YUV 4:2:2, YUV 4:2:0
- Frame rate conversion
- Two possible configurations: Scaler only or full fledged functionality
- Supports resolutions up to 4096 x 2160

The Video Processing Subsystem is a hierarchical IP that bundles a collection of video processing IP subcores and outputs them as a single IP. The video processing IP subsystem has design time configurability in terms of performance and quality. You can configure the subsystem IP through one single graphical user interface. A preview of this GUI is shown in Figure 4-1.

All video processing IP subcores have been developed using Vivado HLS.

Applications

- Color space (RGB/YUV) and format (YUV 4:4:4/4:2:2/4:2:0) conversion.
- Scale up and down up to 4k2k at 60 Hz.
- Zoom mode, where in a user defined window, the input stream is scaled to panel resolution.
- Picture-In-Picture mode where in the input stream is scaled down to a user defined window size and displayed at the user defined co-ordinates on the panel.
- Ability to paint the PIP background to a defined color.
- Interlaced to progressive conversion.
- Frame rate conversion.
  - Drop frames if input rate > output rate.
  - Repeat frames if output rate < input rate.

Licensing and Ordering Information

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado synthesis
- Vivado implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

License Type

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. There is no evaluation version of the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the Video Processing Subsystem product web page.

www.xilinx.com/products/intellectual-property/video-processing-subsystem.html

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 Processing Subsystem core is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. Refer to the *Video IP: AXI Feature Adoption* section of the *Vivado AXI Reference Guide* (UG1037) [Ref 8] for additional information.

Performance

Latency

The latency of the Scaler-only configuration depends on the number of taps used in the vertical scaler. For example, a 6-tap vertical scaler introduces 4 video lines of delay. In full-fledged configuration, the latency depends on the format of the video. In progressive video, the latency is 1 full frame time plus several lines because the video DMA engine is used in the data flow and programmed to read one frame buffer behind the write frame buffer location. In interlaced video, 1 field additional delay is being incurred by the deinterlacer algorithm.

Resource Utilization

Table 2-1 and Table 2-2 show representative resource utilization for the two supported configurations.

Table 2-1: **Resource Utilization for Scaler-only Configuration**

FF	LUTs	BRAMs	DSPs
6,101	7,512	45	72

Table 2-2: **Resource Utilization for Full-fledged Configuration**

FF	LUTs	BRAMs	DSPs
33,538	35,440	138	125

Port Descriptions

Figure 2-1 shows the Video Processing Subsystem IP diagram in its full-fledged configuration. The IP has four AXI interfaces:

- AXI4-Stream streaming video input (`s_axis`)
- AXI4-Stream streaming video output (`m_axis`)
- AXI-MM memory interface (`m_axi_mm`)
- AXI-Lite control interface (`s_axi_ctrl`).

Figure 2-1: **Full-fledged Video Processing Subsystem IP**

The AXI Streaming, AXI Memory, and AXI Lite interfaces can be run at their own clock rate, therefore, three separate clock interfaces are provided named `aclk_axis`, `aclk_axi_mm`, and `aclk_ctrl`, respectively. The `aresetn_ctrl` signal is the reset signal of the IP, and `aresetn_io_axis` is an outgoing signal that can be used to hold IPs in reset when the Video Processing Subsystem IP is not ready to consume data on streaming input. Finally, `deint_field_id` signal indicates field polarity in case of interlaced operation.

AXI4-Stream Video

The video processing IP subsystem has AXI4-Stream video input and output interfaces named `s_axis` and `m_axis`, respectively. These interfaces follow the interface specification as defined in the *Video IP* chapter of the *Vivado AXI Reference Guide* (UG1037) [Ref 8]. 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. For example, the pixel mapping per color format and bus signals for 10 bits per component are shown in Table 2-3 through Table 2-7.

Table 2-3: **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 2-4: **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 2-5: **Dual Pixels per Clock, 10 Bits per Component Mapping for YUV 4:2:2(1)**

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

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

Table 2-6: **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	Y3	U2	Y2	Y1	U0	Y0

Table 2-7: **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	Y2	Y1	V0	Y0

Table 2-8 shows the interface signals for input and output AXI4-Stream video streaming interfaces.

Table 2-8: **AXI4 Streaming Interface Signals**

Name	Direction	Width	Description
s_axis_tdata	In	floor(((3 x bits_per_component x pixels_per_clock) + 7) / 8) x 8	Input Data
s_axis_tready	Out	1	Input Ready
s_axis_tvalid	In	1	Input Valid
s_axis_tdest	In	1	Input data routing identifier
s_axis_tkeep	In	(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	In	1	Input End of Line

Table 2-8: **AXI4 Streaming Interface Signals *(Cont'd)***

Name	Direction	Width	Description
s_axis_tstrb	In	(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	In	1	Input Start of frame
m_axis_tdata	Out	floor(((3 x bits_per_component x pixels_per_clock) + 7) / 8) x 8	Output Data
m_axis_tdest	Out	1	Output data routing identifier
m_axis_tid	Out	1	Output data stream identifier
m_axis_tkeep	Out	(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	Out	1	Output End of Line
m_axis_tready	In	1	Output Ready
m_axis_tstrb	Out	(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	Out	1	Output Start of frame
m_axis_tvalid	Out	1	Output Valid

Both video streaming interfaces run at the video stream clock speed `aclk_axis`.

AXI-MM Memory Interface

The video DMA read and write ports and the deinterlacer read and write ports are concentrated by an AXI-MM cross-bar interconnect such that there is only one AXI-MM interface on the boundary of the subsystem. The AXI-MM interface runs on the `aclk_axi_mm` clock domain. The signals follow the specification as defined in the *AXI Reference Guide* [Ref 8]. The AXI-MM Memory interface is only present with the full-fledged functionality configuration. The Scaler-only configuration does not require access to external memory.

AXI-Lite Control Interface

Table 2-9 shows the AXI-Lite control interface signals. This interface runs at the `aclk_ctrl` clock. Control of the video processing pipe is only supported through the video processing IP subsystem driver.

Table 2-9: **AXI Lite Control Interface**

Name	Direction	Width	Description
s_axi_ctrl_aresetn	In	1	Reset
s_axi_ctrl_aclk	In	1	Clock
s_axi_ctrl_awaddr	In	18	Write address
s_axi_ctrl_awprot	In	3	Write address protection
s_axi_ctrl_awvalid	In	1	Write address valid
s_axi_ctrl_awready	Out	1	Write address ready
s_axi_ctrl_wdata	In	32	Write data
s_axi_ctrl_wstrb	In	4	Write data strobe
s_axi_ctrl_wvalid	In	1	Write data valid
s_axi_ctrl_wready	Out	1	Write data ready
s_axi_ctrl_bresp	Out	2	Write response
s_axi_ctrl_bvalid	Out	1	Write response valid
s_axi_ctrl_bready	In	1	Write response ready
s_axi_ctrl_araddr	In	18	Read address
s_axi_ctrl_arprot	In	3	Read address protection
s_axi_ctrl_arvalid	In	1	Read address valid
s_axi_ctrl_aready	Out	1	Read address ready
s_axi_ctrl_rdata	Out	32	Read data
s_axi_ctrl_rresp	Out	2	Read data response
s_axi_ctrl_rvalid	Out	1	Read data valid
s_axi_ctrl_rready	In	1	Read data ready

Clocks and Resets

Table 2-10 provides an overview of the clocks and resets. See section Clocking in Chapter 3 for more information.

Table 2-10: **Clocks and Resets**

Name	Direction	Width	Description
Clocks			
aclk_axis	In	1	Clock at which AXI4-Stream input and output are running.
aclk_ctrl	In	1	AXI-Lite clock for CPU control interface.
aclk_axi_mm	In	1	Clock at which AXI-MM interface is running.

Table 2-10: **Clocks and Resets *(Cont'd)***

Name	Direction	Width	Description
Resets			
aresetn_ctrl	In	1	Reset, associated with aclk_ctrl (active Low). The aresetn_ctrl signal resets the entire IP including the data path and AXI4-Lite registers.
aresetn_io_axis	Out	1	Used to hold upstream logic in reset while the Video Processing Subsystem is not to consume data on streaming input (active Low).

Field Polarity

The `deint_field_id` signal indicates the polarity of the incoming field when the input video is interlaced. This signal is only used by the deinterlacer with interlaced data. This signal is ignored for progressive video inputs.

Table 2-11: **Field Polarity**

Name	Direction	Width	Description
deint_field_id	In	1	Field polarity, odd is low, high is even

Designing with the Core

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

General Design Guidelines

The Video Processing Subsystem is a collection of individual subcore IPs packaged as a hierarchical IP and configured through one single graphical user interface (GUI). The Subsystem can perform the following functions: deinterlacing, scaling, frame rate conversion, color space conversion and correction, and chroma resampling.

Appendix C, Application Software Development describes how to integrate the associated Video Processing Subsystem API into a software application.

Deinterlacing

The Video Deinterlacer converts live incoming interlaced video streams into progressive video streams. Interlaced images have temporal motion between the two fields that comprise an interlaced frame. The conversion to a progressive format recombines these two fields into one single progressive scan frame. The weaving of interlaced video streams results in unsightly motion artifacts in the progressive output image. For this reason, the Video Deinterlacer uses three field buffers, and produces progressive frames based on a combination of spatial and temporal processing.

Features

The Video Deinterlacer is a low-cost basic Deinterlacer that uses three field buffers. The following list is a summary of the supported features:

- Always uses memory interface.
- Support for RGB, YUV 4:4:4, and YUV 4:2:2.
- 8, 10, 12, or 16 bits per component.

Scaling

Video scaling is the process of converting an input color image of dimensions Xin pixels by Yin lines to an output color image of dimensions Xout pixels by Yout lines. The IP converts a specified rectangular area of an input digital video image from the original sampling grid to a desired target sampling grid.

The input image must be provided in raster scan format (left to right and top to bottom). The valid outputs are also given in this order.

Video scaling is a form of 2-D filter operation which can be approximated with the equation shown in Equation 3-1.

Equation 3-1

$$Pix_{out}(x,y) = \sum_{HTaps-1}^{i=0} \sum_{VTaps-1}^{j=0} Pix_{in}[x - (HTaps/2) + i, y - (VTaps/2) + j] \times Coef(i,j)$$

In this equation, x and y are discrete locations on a common sampling grid; Pix_{out} (x, y) is an output pixel that is being generated at location (x, y); Pix_{in} (x, y) is an input pixel being used as part of the input scaler aperture; Coef (i, j) is an array of coefficients that depend upon the application; and HTaps and VTaps are the number of horizontal and vertical taps in the filter, respectively.

The coefficients in this equation represent weights applied to the set of input samples chosen to contribute to one output pixel, according to the scaling ratio.

Features

The Scaler comes in three different quality levels each at different levels of resource usage.

- Bilinear scaling is the cheapest implementation of the Scaler that uses bilinear interpolation to calculate pixels. Bilinear interpolation produces a greater number of interpolation artifacts (such as aliasing, blurring, and edge halos) than more computationally demanding techniques such as bicubic interpolation.
- Bicubic scaling is a more demanding compared to bilinear scaling, and produces smoother pictures with less artifacts. Compared to bilinear interpolation, which only takes 2 x 2 pixels into account, bicubic interpolation considers a 4 x 4 pixel area.
- The polyphase concept is explained in Polyphase Scaling. The picture quality (and resource usage) of a polyphase scaler depends largely on the number of filter taps used and number of filter phases used. Polyphase scaling offers the highest quality but also has the highest resource utilization.

The following list is a summary of the supported features:

- Polyphase, bicubic, or bilinear scaling modes
- 6, 8, 10, or 12-tap 64 phase polyphase scaling
- Support for RGB, YUV 4:4:4, and YUV 4:2:2.
- 8, 10, 12, or 16 bits per video component

Polyphase Scaling

For scaling, the input and output sampling grids are assumed to be different, in contrast to the example in the preceding section. To express a discrete output pixel in terms of input pixels, it is necessary to know or estimate the location of the output pixel relative to the closest input pixels when superimposing the output sampling grid upon the input sampling grid for the equivalent 2-D space. With this knowledge, the algorithm approximates the output pixel value by using a filter with coefficients weighted accordingly. Filter taps are consecutive data-points drawn from the input image.

As an example, Figure 3-1 shows a desired 5x5 output grid ("O") superimposed upon an original 6x6 input grid ("X"), occupying common space. In this case, estimating for output position (x, y) = (1, 1), shows the input and output pixels to be co-located. You can weigh the coefficients to reflect no bias in either direction, and can even select a unity coefficient set. Output location (2, 2) is offset from the input grid in both vertical and horizontal dimensions. Coefficients can be chosen to reflect this, most likely showing some bias towards input pixel (2, 2), etc. Filter characteristics can be built into the filter coefficients by appropriately applying anti-aliasing low-pass filters.

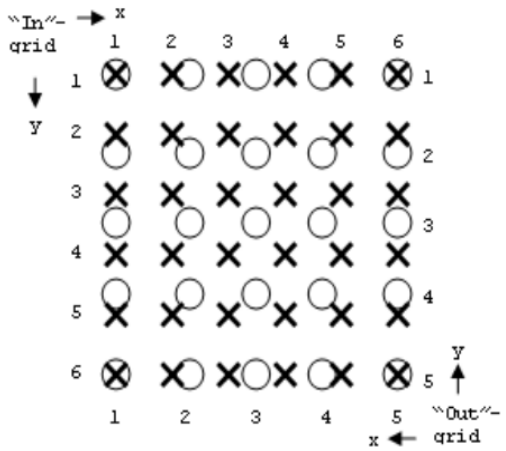

Figure 3-1: **5x5 Output Grid (“O”) Super-imposed over 6x6 Input Grid (“X”)**

The space between two consecutive input pixels in each dimension is conceptually partitioned into a number of bins or phases. The location of any arbitrary output pixel always falls into one of these bins, thus defining the phase of coefficients used. The filter

architecture should be able to accept any of the different phases of coefficients, changing phase on a sample-by-sample basis.

A single dimension is shown in Figure 3-2. As illustrated in this figure, the five output pixels shown from left to right could have the phases 0, 1, 2, 3, 0.

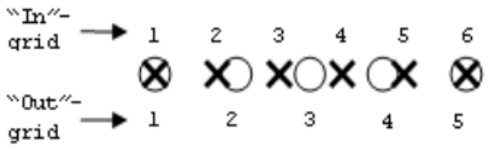

Figure 3-2: **Super-imposed Grids for 1 Dimension**

The examples in Figure 3-1 and Figure 3-2 show a conversion where the ratio Xin/Xout = Yin/Yout = 5/4. This ratio is known as the Scaling Factor, or SF. The horizontal and vertical Scaling Factors can be different. A typical example is drawn from the broadcast industry, where some footage can be shot using 720p (1280 x 720), but the cable operator needs to deliver it as per the broadcast standard 1080p (1920 x 1080). The SF becomes 2/3 in both H and V dimensions.

Typically, when Xin > Xout, this conversion is known as horizontal down-scaling (SF > 1). When Xin < Xout, it is known as horizontal up-scaling (SF < 1).

The set of coefficients constitute filter banks in a polyphase filter whose frequency response is determined by the amount of scaling applied to the input samples. The phases of the filter represent subfilters for the set of samples in the final scaled result.

The number of coefficients and their values are dependent upon the required low-pass, anti-alias response of the scaling filter; for example, smaller scaling ratios require lower passbands and more coefficients. Filter design programs based on the Lanczos algorithm are suitable for coefficient generation. Moreover, MATLAB® product fdatool/fvtool can be used to provide a wider filter design toolset.

A direct implementation of Equation 1 suggests that a filter with VTaps x HTaps multiply operations per output are required. However, the Xilinx® Video Scaler supports only separable filters, which completes an approximation of the 2-D operation using two 1-D stages in sequence - a vertical filter (V-filter) stage and a horizontal filter (H-filter) stage. The intermediate results of the first stage are fed sequentially to the second stage.

The vertical filter stage filters only in the vertical domain, for each incrementing horizontal raster scan position x, creating an intermediate result described as VPix (Equation 3-2).

$$VPix_{int}[x,y] = \sum_{VTaps-1}^{i=0} Pix_{in}[x, y-(VTaps/2)+i] \times Vcoef[i] \qquad \text{Equation 3-2}$$

The output result of the vertical component of the scaler filter is input into the horizontal filter with the appropriate rounding applied. The separation means this can be reduced to the shown VTaps and HTaps multiply operations, saving FPGA resources (Equation 3-3).

$$Pix_{out}[x,y] = \sum_{HTaps-1}^{i=0} VPix_{int}[x-(HTaps/2)+i,y] \times Hcoef[i] \qquad \textit{Equation 3-3}$$

Notice that the differences between the Bilinear, Bicubic, and Polyphase architectures are not only marked by a difference in coefficients only but with the implementation of optimized architectures for Bilinear and Bicubic scaling.

Video DMA Engine

Many video applications require frame buffers to handle frame rate changes or changes to the image dimensions (scaling or cropping). The Video DMA engine, which uses the Xilinx AXI Video Direct Memory Access IP, is designed to allow for efficient high-bandwidth access between AXI4-Stream video interface and AXI4-MM interface.

Features

The Video DMA engine within the full-fledged configuration of the Video Processing Subsystem IP is used to enable frame rate conversion and also more advanced scaling use cases like crop and zoom and scale to picture in picture.

Frame Rate Conversion

Frame rate conversion is implemented by dropping or repeating frames. The DMA engine keeps track of 4 frame buffers that are being written to in a cyclic fashion. The read portion of the video data flow remains exactly 1 frame behind the write pointer. In case the incoming frame rate is higher than the outgoing frame rate, the write pointer advances more frequently than the read pointer, meaning that frames are skipped. Similarly, the read pointer is always one frame behind the write pointer even in case the outgoing frame rate is higher than the incoming frame rate. In this case, frames are repeated.

Crop and Zoom

Another use of the video DMA engine is to enable advanced scaling use cases like crop and zoom, and scale to picture in picture. Without the use of memory, only basic scaling can be performed from one resolution to another resolution, as shown in Figure 3-3.

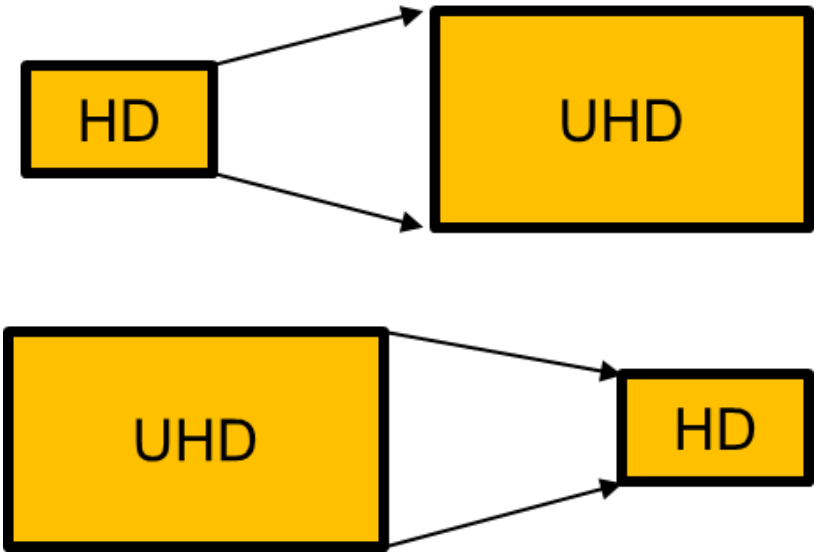

Figure 3-3: **Basic Scaling**

With the video DMA engine in the video path, it is possible to enable a crop and zoom feature as shown in Figure 3-4 by cropping out of memory.

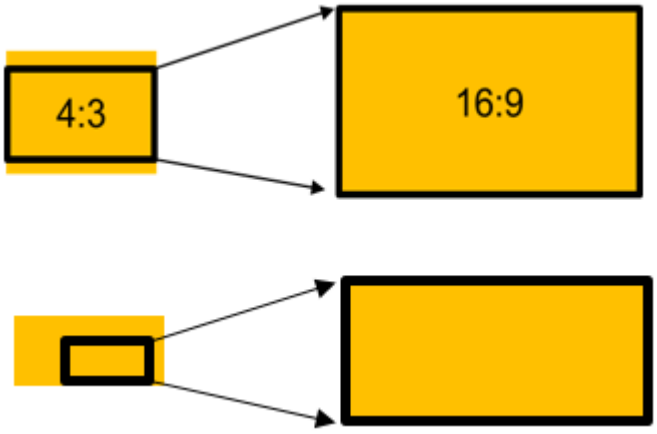

Figure 3-4: **Crop and Zoom**

Picture in Picture

Alternatively, it is also possible to enable a picture in picture feature as shown in Figure 3-5.

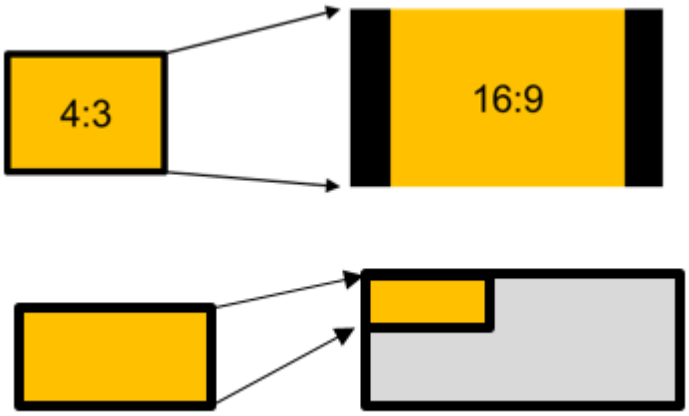

Figure 3-5: **Picture in Picture**

Note that there is no data re-alignment engine implemented, therefore, memory access is aligned to the granularity of the bus. This is automatically taken care of by the driver.

Memory Requirement

DDR memory is used to store video frame buffers for full configuration mode. Subsystem uses 5 frame buffers for Progressive input and 3 field buffers for interlaced input. You can calculate the amount of memory required by the subsystem using the following equations:

5 * MAX_WIDTHp * MAX_HEIGHTp * NUM_VIDEO_COMPONENTS * BytesPerComp

+

3 * MAX_WIDTHi * MAX_HEIGHTi * NUM_VIDEO_COMPONENTS * BytesPerComp

BytesPerComp

- 1 Byte for 8 bit data pipe
- 2 Byte for 10/12/16 bit data pipe

Memory Bandwidth

You require the following Memory Bandwidth requirements with full-fledged configuration:

The deinterlacer writes 1 field and reads 2 fields. For example, a resolution of 1080i of 8 bit RGB data at 60 Hz:

write 1 field = 1920 columns * 540 rows * 24bits * 60 fps = 178MBytes/second

read 2 fields = 1920 columns * (2*540) rows * 24bits * 60 fps = 356MBytes/second

The VDMA writes 1 frame and reads 1 frame. For example, a 4K resolution of 8 bit RGB data at 60 Hz:

write 1 frame = 3840 columns * 2160 rows * 24 bits * 60 fps = 1424MBytes/second

read 1 frame = 3840 columns * 2160 rows * 24 bits * 60 fps = 1424MBytes/second

Color Space Conversion and Correction

There are many variations that cause difficulties in accurately reproducing color in imaging systems. These can include:

- Spectral characteristics of the optics (lens, filters)
- Lighting source variations like daylight, fluorescent, or tungsten
- Characteristics of the color filters of the sensor

The Color Space Converter/Correction function provides a method for correcting the image data for these variations. This fundamental block operates on either YUV or RGB data.

As an example, following one of the three color channels through an imaging system from the original light source to the processed image helps understand the functionality of this core.

The blue color channel is a combination of the blue photons from the scene, multiplied by the relative response of the blue filter, multiplied by the relative response of the silicon to blue photons. However, the filter and silicon responses might be quite different from the response of the human eye, so blue to the sensor is quite different from blue to a human being.

This difference can be corrected and made to more closely match the blue that is acceptable to human vision. The Color Space Converter/Correction function multiplies the pixel values by some coefficient to strengthen or weaken it, creating an effective gain. At the same time a mixture of green or red can be added to the blue channel. To express this processing mathematically, the new blue (Bc) is related to the old blue (B), red (R), and green (G) according to:

$$Bc = K1 \times R + K2 \times G + K3 \times B \qquad \textit{Equation 3-4}$$

where K1, K2, and K3 are the weights for each of the mix of red, green, and blue to the new blue.

Extending this concept, a standard 3 x 3 matrix multiplication can be applied to each of the color channels in parallel simultaneously. This is a matrix operation where the weights define a color-correction matrix. In typical applications, color-correction also contains offset compensation to ensure black [0,0,0] levels are achieved.

$$\begin{bmatrix} R_c \\ G_c \\ B_c \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} \quad \text{Equation 1-1}$$

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} + \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} \quad \text{Equation 1-2}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} + \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} \quad \text{Equation 1-3}$$

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} \quad \text{Equation 1-4}$$

Figure 3-6: **Various Color Space Conversions**

As shown in the matrix operation, the input pixels are transformed to a set of corrected output pixels. This can be a very useful function configured as a static application; however, the programmability of the coefficients and offset values allows this function to adapt to changing lighting conditions based on a separate control loop.

Features

The Color Space Conversion and Correction function offers a 3 x 3 matrix multiplication for a variety of color correction applications. The coefficient matrix is fully programmable and includes offset compensation, and clipping and clamping of the output is also definable.

The following list is a summary of the supported features:

- User programmable matrix coefficients
- Support for RGB, YUV 4:4:4, and YUV 4:2:2
- 8, 10, 12, or 16 bits per component
- Driver API to set coefficients for converting RGB to YUV, or vice-versa
- Driver API to set/get brightness, contrast, saturation and gain

Color Space Conversion

The primary purpose of this function is to provide color space conversion between the RGB and YUV domains. The fully programmable 3 x 3 matrix with offsets and clipping and clamping allow the support of multiple video standards.

Color Correction

This function provides support for additional color correction within a user defined window in the video frame. You can define a second coefficient matrix to be applied only within a demo window. You also program the size and position of the demo window.

Filter Coefficients and Offsets

The coefficients are presented in 16.12 fixed point format. The 16-bit signed integer values (2's compliment) are equivalent to real numbers in the [-8 .. 8] range.

The offset value has a width of the output data width plus 1. It is a signed integer with a range as shown in Equation 3-5.

$$[-2^{Out\_Data\_Width}, 2^{Out\_Data\_Width}-1] \quad \textit{Equation 3-5}$$

Output values greater than the Clip value are replaced with the Clip value. Output values smaller than the Clamp value are replaced with the Clamp value. The Clip and Clamp values have the same width as the output data width. They are unsigned integers with a range of $[0 .. 2^{Out\_Data\_Width}-1]$.

Matrix computation outputs are rounded to DATA_WIDTH bits by adding half an output LSB prior to truncation.

Chroma Resampling

The human eye is not as receptive to chrominance (color) detail as luminance (brightness) detail. Using color-space conversion, it is possible to convert RGB into the YUV color space, where Y is Luminance information, and U and V are derived color difference signals. At normal viewing distances, there is no perceptible loss incurred by sampling the color difference signals (U and V) at a lower rate to provide a simple and effective video compression to reduce storage and transmission costs

The Chroma Resampler function converts between chroma sub-sampling formats of 4:4:4, 4:2:2, and 4:2:0. There are a total of six conversions available for the three supported sub-sampling formats. Conversion is achieved using a FIR filter approach. Some conversions require filtering in only the horizontal dimension, only the vertical dimension, or both. Interpolation operations are implemented using a two-phase polyphase FIR filter. Decimation operations are implemented using a low-pass two-phase polyphase FIR filter to suppress chroma aliasing.

Features

The Chroma Resampler function converts between different chroma sub-sampling formats. The supported formats are 4:4:4, 4:2:2, and 4:2:0. There are three different options for interpolating and decimating the video samples:

- Define a configurable filter with programmable coefficients for high-performance applications.
- Use the pre-defined static filter with power-of -two coefficients for low-footprint applications.
- Replicate or drop pixels for minimal footprint.

The following list is a summary of the supported features:

- User programmable filter coefficients
- 4, 6, 8, or 10 tap filter
- Support for RGB, YUV 4:4:4, and YUV 4:2:2, and YUV 4:2:0.
- 8, 10, 12, or 16 bits per component

Sub-sampled Video Formats

The sub-sampling scheme is commonly expressed as a three part ratio J:a:b (for example, 4:2:2), that describes the number of luminance and chrominance samples in a conceptual region that is J pixels wide, and 2 pixels high. The parts are (in their respective order):

- J: Horizontal sampling reference (width of the conceptual region). This is usually 4.
- a: Number of chrominance samples (V, U) in the first row of J pixels.
- b: Number of (additional) chrominance samples (V, U) in the second row of J pixels.

To illustrate the most common sub-sampling schemes, Figure 3-7 introduces a graphical notation of sampling grid pixels.

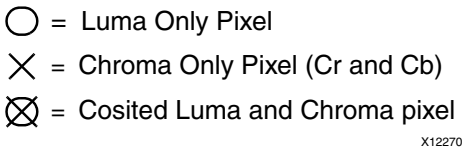

Figure 3-7: **Luma, Chroma Notation**

4:4:4

Similar to RGB, the 4:4:4 format is used for image capture and display purposes. U and V channels are sampled at the same rate as luminance. Hence, all pixel locations have luma and chroma data co-sited, as shown in Figure 3-8.

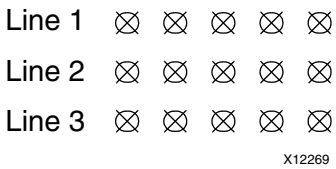

Figure 3-8: **YUV 4:4:4 Format**

4:2:2

This format contains horizontally sub-sampled chroma. For every two luma samples, there is an associated pair of U and V samples. The sub-sampled chroma locations are co-sited with alternate luma samples as shown in Figure 3-9.

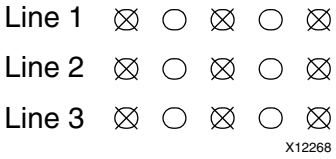

Figure 3-9: **YUV 4:2:2 Format**

4:2:0

4:2:0 encoding contains horizontally and vertically sub-sampled chroma. Horizontal and vertical chroma positions are co-sited with alternate luma samples on alternate scanlines. The sampling positions are shown in Figure 3-10.