Video and Image Processing Suite User Guide

Updated for Intel® Quartus® Prime Design Suite: **22.1**

Online Version

Send Feedback

UG-VIPSUITE

ID: **683416**

Version: **2022.09.29**

Contents

1. About the Video and Image Processing Suite

The Intel® Video and Image Processing (VIP) Suite is available in the DSP library of the Intel Quartus® Prime software. You can configure the IPs to the required number of bits per symbols, symbols per pixel, symbols in sequence or parallel, and pixels in parallel.

The VIP Suite offers the following IPs:

- 2D FIR II Intel FPGA IP
- Avalon-ST Video Monitor Intel FPGA IP (Available only in Platform Designer (Standard) edition)
- Avalon-ST Video Stream Cleaner Intel FPGA IP
- Chroma Resampler II Intel FPGA IP
- Clipper II Intel FPGA IP
- Clocked Video Input II Intel FPGA IP
- Clocked Video Output II Intel FPGA IP
- Color Plane Sequencer II Intel FPGA IP
- Color Space Converter II Intel FPGA IP
- Configurable Guard Bands Intel FPGA IP
- Control Synchronizer Intel FPGA IP (Available only in Platform Designer (Standard) edition)
- Deinterlacer II Intel FPGA IP
- Frame Buffer II Intel FPGA IP
- Gamma Corrector II Intel FPGA IP
- Interlacer II Intel FPGA IP
- Mixer II Intel FPGA IP
- Scaler II Intel FPGA IP
- Switch II Intel FPGA IP
- Test Pattern Generator II Intel FPGA IP
- Trace System Intel FPGA IP (Available only in Platform Designer (Standard) edition)
- Warp Lite IP

These IPs transmit and receive video according to the Avalon streaming video standard. Most IPs receive and transmit video data according to the same Avalon streaming video configuration, but some explicitly convert from one Avalon streaming video configuration to another. For example, you can use the Color Plane Sequencer II IP to convert from 1 pixel in parallel to 4.

All VIP IPs require even frame widths when using 4:2:2 data; odd frame widths create unpredictable results or distorted images. The Clipper II IP requires even clip start offsets and the Mixer II IP requires even offsets when using 4:2:2 data.

The signal names are standard Avalon streaming signals, and so by default, not enumerated. Some IPs may have additional signals.

All IPs in the VIP Suite support pixels in parallel, with the exception of Control Synchronizer, and Avalon-ST Video Monitor IP. Most of the IPs support 8 pixels in parallel and 8K resolutions.

Table 1. Minimum and Maximum Supported X and Y Resolutions

VIP IP Cores	Minimum Input and Output Resolution in Pixels (Width × Height)	Maximum Input and Output Resolution in Pixels (Width × Height)
Clocked Video Input II	32 × 32	8192 × 8192
Clocked Video Output II	32 × 32	8192 × 8192
Control Synchronizer	32 × 32	1920 × 1080
Deinterlacer II	32 × 32	4096 × 2160 (1)
Frame Buffer II (Frame Reader and Frame Writer)	32 × 32	7680 × 4320
Warp Lite IP	128 × 128	1920 × 1080
Other IP	32 × 32	8192 × 8192

Related Information

Video and Image Processing Suite User Guide Archives on page 229
Provides a list of user guides for previous versions of the Video and Image Processing Suite IP cores.

1.1. Release Information

Release information for the Video and Image Processing Suite.

Intel FPGA IP versions match the Intel Quartus Prime Design Suite software versions until v19.1. Starting in Intel Quartus Prime Design Suite software version 19.2, Intel FPGA IP has a new versioning scheme.

The Intel FPGA IP version (X.Y.Z) number can change with each Intel Quartus Prime software version. A change in:

- X indicates a major revision of the IP. If you update the Intel Quartus Prime software, you must regenerate the IP.
- Y indicates the IP includes new features. Regenerate your IP to include these new features.
- Z indicates the IP includes minor changes. Regenerate your IP to include these changes.

(1) Deinterlacer II can pass through progressive video up to a maximum resolution of 4096 × 2160.

Table 2. Release Information

Item	Description
Version	22.1
Release Date	March 2022
Ordering Code	IPS-VIDEO (Video and Image Processing Suite)

Intel verifies that the current version of the Intel Quartus Prime software compiles the previous version of each IP core, if this IP core was included in the previous release. Intel reports any exceptions to this verification in the *Intel FPGA IP Release Notes*. Intel does not verify compilation with IP core versions older than the previous release.

Related Information

- Intel FPGA IP Library Release Notes
- Errata for VIP Suite in the Knowledge Base

1.2. Device Family Support

Table 3. Device Family Support

The table lists the device support information for the Video and Image Processing Suite IP cores. The Warp Lite IP only supports Intel Arria® 10 and Intel Stratix® 10 devices.

Device Family	Support
Intel Stratix 10	Final
Intel Agilex™	Final (2)
Intel Arria 10	Final
Intel Cyclone® 10 LP	Final
Intel Cyclone 10 GX	Final
Intel MAX® 10	Final
Arria II GX/GZ	Final
Arria V GX/GT/SX/ST	Final
Arria V GZ	Final
Cyclone IV E/GX	Final
Cyclone V	Final
Stratix IV	Final
Stratix V	Final
Other device families	No support

(2) The following IPs don't support Intel Agilex devices: Deinterlacer II , Frame Buffer II, Clocked Video Interface.

The following terms define device support levels for Intel FPGA IP cores:

- Advance support—the IP core is available for simulation and compilation for this device family. Timing models include initial engineering estimates of delays based on early post-layout information. The timing models are subject to change as silicon testing improves the correlation between the actual silicon and the timing models. You can use this IP core for system architecture and resource utilization studies, simulation, pinout, system latency assessments, basic timing assessments (pipeline budgeting), and I/O transfer strategy (data-path width, burst depth, I/O standards tradeoffs).
- Preliminary support—the IP core is verified with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.
- Final support—the IP core is verified with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

1.3. Latency

You can use the latency information to predict the approximate latency between the input and the output of your video processing pipeline.

The latency is described using one or more of the following measures:

- the number of progressive frames
- the number of interlaced fields
- the number of lines when less than a field of latency
- a number of cycles

When you measure the latency, assume that other functions on the data path are not stalling the IP; (the output ready signal is high).

Table 4. Video and Image Processing Suite Latency

The table lists the approximate latency from the video data input to the video data output for typical usage modes of the Video and Image Processing IPs.

IP Core	Mode	Latency
2D FIR Filter Latency	Filter size: N × N	(*N*–1) lines + (*n* cycles)
Mixer II	All modes	*n* cycles
Avalon-ST Video Stream Cleaner	All modes	*n* cycles
Chroma Resampler II	Input format: 4:2:2; Output format: 4:4:4	*n* cycles
	Input format: 4:2:0; Output format: 4:4:4 or 4:2:2	1 line + *n* cycles
Clipper II	All modes	*n* cycles
Clocked Video Input II	All modes	*n* cycles
Clocked Video Output II	All modes	*n* cycles
Color Plane Sequencer II	All modes	*n* cycles
Color Space Converter II	All modes	*n* cycles

continued...

IP Core	Mode	Latency
Control Synchronizer	All modes	*n* cycles
Deinterlacer II	• Method: Bob • Frame buffering: None	*n* cycles
	• Method: Weave • Frame buffering: None	1 field + *n* cycles
	• Method: Motion-adaptive • Frame buffering: None • Output frame rate: As input field rate	2 lines + *n* cycles
	• Method: Motion-adaptive, video-over-film mode • Frame buffering: 3 input fields are buffered • Output frame rate: same as the input field rate	1 field + 2 lines, or 2 lines 40% to 60% (depending on phasing) of the time, the core performs a weave forward so there is no initial field of latency.
Frame Buffer II	All modes	1 frame + *n* cycles
Gamma Corrector II	All modes	*n* cycles
Interlacer II	All modes	*n* cycles
Scaler II	• Scaling algorithm: Polyphase • Number of vertical taps: N	(*N* –1) lines + *n* cycles
Switch II	All modes	*n* cycles
Test Pattern Generator II	Not applicable.	
Warp Lite IP	All modes	Up to 100 lines. The amount varies across the frame depending on the level of distortion specified.

1.4. In-System Performance and Resource Guidance

The performance and resource data provided for your guidance.

Note: Run your own synthesis and f_{MAX} trials to confirm the listed IP cores meet your system requirements.

Table 5. Performance and Resource Data Using Intel Arria 10 Devices

The following data are obtained through a 4K test design example using an Intel Arria 10 device (10AX115S3F45E2SGE3).

The general settings for the design is 10 bits per color plane; 2 pixels in parallel. The target f_{MAX} is 300 MHz.

IP Core	Configuration	ALMs	M20K	DSP Blocks
Mixer II	• Number of color planes in parallel = 3 • Inputs = 2 • Output = 1 • Internal Test Pattern Generator	1,890	0	12
Clocked Video Input II	• Number of color planes in parallel = 3 • Sync signals = On separate wires • Pixel FIFO size = 4096 pixels • Use control port = On	855	14	0

continued...

IP Core	Configuration	ALMs	M20K	DSP Blocks
Clocked Video Output II	• Number of color planes in parallel = 3 • Sync signals = On separate wires • Pixel FIFO size = 4096 pixels • Use control port = On • Run-time configurable video modes = 4	2,251	12	0
Color Space Converter II	• Run-time control = On • Color model conversion = RGB to YCbCr	865	1	12
Frame Buffer II	• Maximum frame size = 4096 × 2160 • Number of color planes in parallel = 3 • Avalon-MM master ports width = 512 • Read/write FIFO depth = 32 • Frame dropping = On • Frame repeating = On	2,232	33	2
Clipper II	• Number of color planes in parallel = 3 • Enable run-time control of clipping parameters = On	963	0	0
Warp Lite	• Bits per symbol = 10 • Maximum width = 1920 • Maximum distortion = 30 • Paired with VFB = 0	6,530	395	21

1.5. Stall Behavior and Error Recovery

The Video and Image Processing Suite IPs do not continuously process data. Instead, they use flow-controlled Avalon streaming interfaces, which allow them to stall the data while they perform internal calculations.

During control packet processing, the IPs might stall frequently and read or write less than once per clock cycle. During data processing, the IPs generally process one input or output per clock cycle. The IPs have some stalling cycles. Typically, the stalling cycles are for internal calculations between rows of image data and between frames/fields.

When stalled, an IP indicates that it is not ready to receive or produce data. The time spent in the stalled state varies between IPs and their parameterizations. In general, it is a few cycles between rows and a few more between frames.

If data is not available at the input when required, all of the IPs stall and do not output data. With the exceptions of the Deinterlacer and Frame Buffer in double or triple-buffering mode, none of the IPs overlap the processing of consecutive frames. The first sample of frame F + 1 is not input until after the IPs produce the last sample of frame F.

When the IPs receive an `endofpacket` signal unexpectedly (early or late), the IPs recover from the error and prepare for the next valid packet (control or data).

IP Core	Stall Behavior	Error Recovery
2D FIR Filter II	• Has a delay of a little more than N–1 lines between data input and output in the case of a *N*×*N* 2D FIR Filter. • Delay caused by line buffering internal to the IP.	An error condition occurs if an `endofpacket` signal is received too early or too late for the run-time frame size. In either case, the 2D FIR Filter always creates output video packets of the configured size.

continued...

IP Core	Stall Behavior	Error Recovery
		• If an input video packet has a late `endofpacket` signal, then the extra data is discarded. • If an input video packet has an early `endofpacket` signal, then the video frame is padded with an undefined combination of the last input pixels.
Mixer II	All modes stall for a few cycles after each output frame and between output lines. Between frames, the IP processes non-image data packets from its input layers in sequential order. The IP may exert backpressure during the process until the image data header has been received for all its input. During the mixing of a frame, the IP: • Reads from the background input for each non-stalled cycle. • Reads from the input ports associated with layers that currently cover the background image. Because of pipelining, the foreground pixel of layer N is read approximately N active cycles after the corresponding background pixel has been read. • If the output is applying backpressure or if one input is stalling, the pipeline stalls and the backpressure propagates to all active inputs. • When alpha blending is enabled, one data sample is read from each alpha port once each time that a whole pixel of data is read from the corresponding input port. There is no internal buffering in the IP, so the delay from input to output is just a few clock cycles and increases linearly with the number of inputs.	The Mixer II IP core processes video packets from the background layer until the end of packet is received. • Receiving an `endofpacket` signal too early for the background layer—the IP core enters error mode and continues writing data until it has reached the end of the current line. The `endofpacket` signal is then set with the last pixel sent. • Receiving an `endofpacket` signal early for one of the foreground layers or for one of the alpha layers—the IP core stops pulling data out of the corresponding input and pads the incomplete frame with undefined samples. • Receiving an `endofpacket` signal late for the background layer, one or more foreground layers, or one or more alpha layers—the IP core enters error mode. This error recovery process maintains the synchronization between all the inputs and is started once the output frame is completed. A large number of samples may have to be discarded during the operation and backpressure can be applied for a long time on most input layers. Consequently, this error recovery mechanism could trigger an overflow at the input of the system.
Avalon-ST Video Stream Cleaner	All modes stall for a few cycles between frames and between lines.	• Receiving an early `endofpacket` signal—the IP core stalls its input but continues writing data until it has sent an entire frame. • Not receiving an `endofpacket` signal at the end of a frame—the IP core discards data until it finds end-of-packet.
Chroma Resampler II	All modes stall for a few cycles between frames and between lines. Latency from input to output varies depending on the operation mode of the IP core. • The only modes with latency of more than a few cycles are 4:2:0 to 4:2:2 and 4:2:0 to 4:4:4—corresponding to one line of 4:2:0 data • The quantities of data input and output are not equal because this is a rate-changing function. • Always produces the same number of lines that it accepts—but the number of samples in each line varies according to the subsampling pattern used. When not stalled, always processes one sample from the more fully sampled side on each clock cycle. For example, the subsampled side	• Receiving an early `endofpacket` signal—the IP core stalls its input but continues writing data until it has sent an entire frame. • Not receiving an `endofpacket` signal at the end of a frame—the IP core discards data until it finds end-of-packet.

continued...

IP Core	Stall Behavior	Error Recovery
	pauses for one third of the clock cycles in the 4:2:2 case or half of the clock cycles in the 4:2:0 case.	
Clipper II	• Stalls for a few cycles between lines and between frames. • Internal latency is less than 10 cycles. • During the processing of a line, it reads continuously but only writes when inside the active picture area as defined by the clipping window.	• Receiving an early `endofpacket` signal—the IP core stalls its input but continues writing data until it has sent an entire frame. • Not receiving an `endofpacket` signal at the end of a frame—the IP core discards data until it finds end of packet.
Clocked Video Input II (3)	• Dictated by incoming video. • If its output FIFO is empty, during horizontal and vertical blanking periods the IP core does not produce any video data.	If an overflow is caused by a downstream core failing to receive data at the rate of the incoming video, the Clocked Video Input sends an `endofpacket` signal and restart sending video data at the start of the next frame or field.
Clocked Video Output II	• Dictated by outgoing video. • If its input FIFO is full, during horizontal and vertical blanking periods the IP stalls and does not take in any more video data.	• Receiving an early `endofpacket` signal—the IP resynchronizes the outgoing video data to the incoming video data on the next start of packet it receives. • Receiving a late `endofpacket`— the IP resynchronizes the outgoing video data to the incoming video immediately.
Color Plane Sequencer II	• Stalls for a few cycles between frames and user/control packets • The Avalon-ST Video transmission settings (color planes in sequence/parallel, number of color planes and number of pixels per beat) determine the throughput for each I/O. The slowest interface limits the overall rate of the others	• Processes video packets until the IP core receives an `endofpacket` signal on either inputs. Frame dimensions taken from the control packets are not used to validate the sizes of the input frames.. • When receiving an `endofpacket` signal on either `din0` or `din1`; the IP core terminates the current output frame. • When both inputs are enabled and the `endofpacket` signals do not line up, extra input data on the second input is discarded until the end of packet is signaled.
Color Space Converter II	• Only stalls between frames and not between rows. • It has no internal buffering apart from the registers of its processing pipeline—only a few clock cycles of latency.	• Processes video packets until the IP core receives an `endofpacket` signal—the control packets are not used. • Any mismatch of the `endofpacket` signal and the frame size is propagated unchanged to the next IP core.
Control Synchronizer	• Stalls for several cycles between packets. • Stalls when it enters a triggered state while it writes to the Avalon-MM Slave ports of other IP cores. • If the slaves do not provide a wait request signal, the stall lasts for no more than 50 clock cycles. Otherwise the stall is of unknown length.	• Processes video packets until the IP core receives an `endofpacket` signal—the image width, height and interlaced fields of the control data packets are not compared against the following video data packet. • Any mismatch of the `endofpacket` signal and the frame size of video data packet is propagated unchanged to the next IP core.

continued...

(3) For CVI II IP, the error recovery behavior varies depending on the Platform Designer parameters. Refer to the *Clocked Video Interface IP* for more information.

IP Core	Stall Behavior	Error Recovery
Deinterlacer II	Stores input video fields in the external memory and concurrently uses these input video fields to construct deinterlaced frames. • Stalls up to 50 clock cycles for the first output frame. • Additional delay of one line for second output frame because the IP core generates the last line of the output frame before accepting the first line of the next input field. • Delay of two lines for the following output frames, which includes the one line delay from the second output frame. • For all subsequent fields, the delay alternates between one and two lines.	• Bob and Weave configurations always recover from an error caused by illegal control or video packets. • Motion adaptive modes require the embedded stream cleaner to be enabled to fully recover from errors.
Frame Buffer II	• May stall frequently and read or write less than once per clock cycle during control packet processing. • During data processing at the input or at the output, the stall behavior of the IP core is largely decided by contention on the memory bus.	• Does not rely on the content of the control packets to determine the size of the image data packets. • Any early or late `endofpacket` signal and any mismatch between the size of the image data packet and the content of the control packet are propagated unchanged to the next IP core. • Does not write outside the memory allocated for each non-image and image Avalon-ST video packet—packets are truncated if they are larger than the maximum size defined at compile time.
Gamma Corrector II	• Stalls only between frames and not between rows. • Has no internal buffering aside from the registers of its processing pipeline— only a few clock cycles of latency	• Processes video packets until the IP core receives an `endofpacket` signal—non-image packets are propagated but the content of control packets is ignored. • Any mismatch of the `endofpacket` signal and the frame size is propagated unchanged to the next IP core.
Interlacer II	• Alternates between propagating and discarding a row from the input port while producing an interlaced output field—the output port is inactive every other row. • The delay from input to output is a few clock cycles when pixels are propagated.	• Receiving `endofpacket` signal later than expected—discards extra data. • Receiving an early `endofpacket` signal—the current output field is interrupted as soon as possible and may be padded with a single undefined pixel.

continued...

IP Core	Stall Behavior	Error Recovery
Scaler II	• The ratio of reads to writes is proportional to the scaling ratio and occurs on both a per-pixel and a per-line basis. • The frequency of lines where reads and writes occur is proportional to the vertical scaling ratio. • For example, scaling up vertically by a factor of 2 results in the input being stalled every other line for the length of time it takes to write one line of output; scaling down vertically by a factor of 2 results in the output being stalled every other line for the length of time it takes to read one line of input. • In a line that has both input and output active, the ratio of reads and writes is proportional to the horizontal scaling ratio. For example, scaling from 64×64 to 128×128 causes 128 lines of output, where only 64 of these lines have any reads in them. For each of these 64 lines, there are two writes to every read. The internal latency of the IP core depends on the scaling algorithm and whether any run time control is enabled. The scaling algorithm impacts stalling as follows: • Bilinear mode: a complete line of input is read into a buffer before any output is produced. At the end of a frame there are no reads as this buffer is drained. The exact number of possible writes during this time depends on the scaling ratio. • Polyphase mode with N_v vertical taps: N_v – 1 lines of input are read into line buffers before any output is ready. The scaling ratio depends on the time at the end of a frame where no reads are required as the buffers are drained. Enabling run-time control of resolutions affects stalling between frames: • With no run-time control: about 10 cycles of delay before the stall behavior begins, and about 20 cycles of further stalling between each output line. • With run-time control of resolutions: about additional 25 cycles of delay between frames.	• Receiving an early `endofpacket` signal at the end of an input line—the IP core stalls its input but continues writing data until it has sent one further output line. • Receiving an early `endofpacket` signal part way through an input line—the IP core stalls its input for as long as it takes for the open input line to complete; completing any output line that may accompany that input line. Then continues to stall the input, and writes one further output line. • Not receiving an `endofpacket` signal at the end of a frame—the IP core discards extra data until it finds an end of packet.
Switch II	• Only stalls its inputs when performing an output switch. • Before switching its outputs, the IP core synchronizes all its inputs and the inputs may be stalled during this synchronization.	—
Test Pattern Generator II	• All modes stall for a few cycles after a field control packet, and between lines. • When producing a line of image data, the IP core produces one sample output on every clock cycle, but it can be stalled without consequences if other functions down the data path are not ready and exert backpressure.	—
Warp Lite	All modes produce a few cycles of stall if the line store fills.	The Warp Lite IP has a built in stream cleaner to handle error conditions.

IP Core	Stall Behavior	Error Recovery
	The line store size ensures it shouldn't fill unless a mismatch occurs between input and output rates.	Receiving an `endofpacket` signal too early causes the stream cleaner to pad the packet up to the frame size specified by the preceding control packet. Receiving an `endofpacket` signal too late causes the stream cleaner to clip the packet to the frame size specified by the preceding control packet. Receiving a video packet without a preceding control packet causes the stream cleaner to discard the video packet.

2. Avalon Streaming Video

The Video and Image Processing IPs conform to the Avalon streaming video standard of data transmission.

This standard is a configurable protocol layer that sits on top of the Intel Avalon streaming standard. The standard comprises video packets, control packets, and/or user packets.

Note: Before you start using the VIP IPs, you must fully understand this protocol layer. The IPs transmit and receive all video data in this format.

The individual video formats supported (i.e. NTSC, 1080p, UHD 4K) depend primarily on the configuration of the Avalon streaming video standard and the clock frequency. The IPs may transmit pixel information either in sequence or in parallel, in RGB or YCbCr color spaces, and under a variety of different chroma samplings and bit depths, depending on which is the most suitable for the end application. The Avalon streaming video protocol adheres to the Avalon streaming standard packet data transfers, with backpressure and a ready latency of 1.

Figure 1. Avalon Streaming Video Signals

The figure shows two VIP IPs and the Avalon streaming video signals used for data transfer. The Avalon streaming optional `channel` signal is always unused.

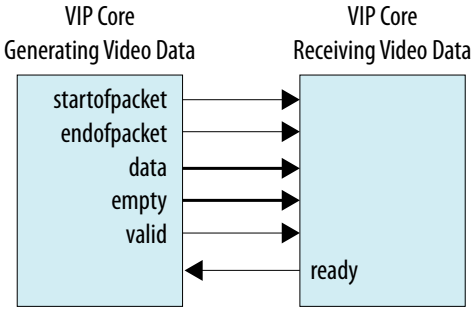
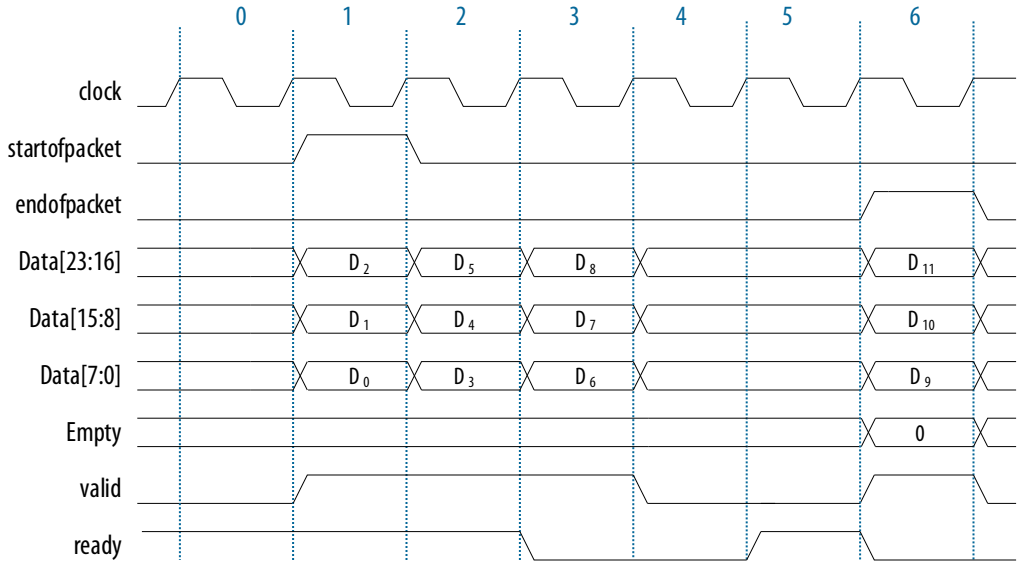


Figure 2. Avalon Streaming Video Packet Transmission (Symbols in Parallel)

The figure shows an example transmission of twelve "symbols" using the Avalon streaming video configuration of the Avalon streaming specification.



A "ready latency" of 1 is used for Avalon streaming video. The example shows the receiving video sink drops its `ready` signal in cycle 3, to indicate that it is not ready to receive any data in cycles 4 or 5. The video source responds by extending its `valid`, `endofpacket` and `data` signals into cycle 6. As the `ready` signal returns high in cycle 5, the video source data in cycle 6 is safely registered by the sink.

The symbols D0, D1… can be pixel color plane data from an Avalon streaming video image packet or data from a control packet or a user packet. The type of packet is determined by the lowest 4 bits of the first symbol transmitted.

Table 6. Avalon Streaming Packet Type Identifiers

Type Identifier D0[3:0]	Description
0x0 (0)	Video data packet
0x1–0x8 (1–8)	User data packet
0x9–0xC (9–12)	Reserved
0xD (13)	Clocked Video data ancillary user packet
0xE (14)	Reserved
0xF (15)	Control packet

Related Information

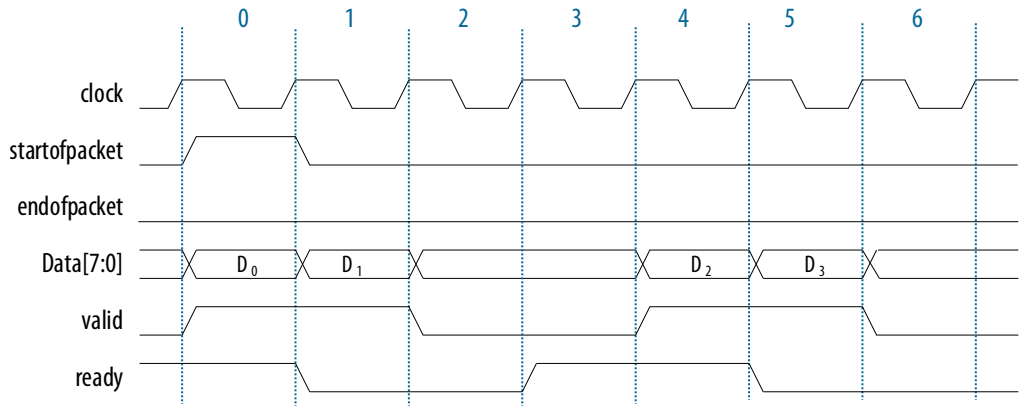
Avalon Interface Specifications
Provides more information about these interface types.

2.1. Avalon-ST Video Configuration Types

The Avalon-ST video protocol also allows for symbols to be transmitted in sequence. The start of the same transaction transmitted in "symbols in sequence" configuration.The symbols themselves are unchanged, but are transmitted consecutively rather than being grouped together.

Most color spaces require more than one symbol to represent each pixel. For example, three symbols are needed for each of the red, green and blue planes of the RGB color space.

Figure 3. Avalon-ST Video Packet Transmission (Symbols in Series)

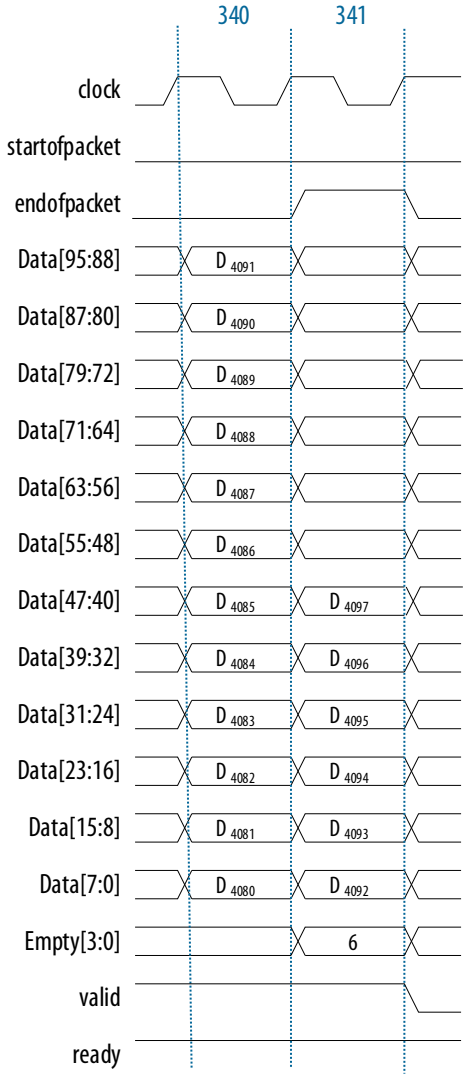


Avalon-ST Video allows for multiple pixels to be transmitted in parallel. When the number of pixels transmitted in parallel is greater than one, the optional Avalon-ST empty signal is added to the data transfer interface between the IP cores.

The figure below shows 4 pixels, each comprising 3 symbols (or color planes), being transmitted in parallel.

Figure 4. Avalon-ST Video Packet Transmission (Pixels in Parallel)

This figure illustrates the end of a video packet containing 1366 pixels, where the final beat of the transaction is not fully populated with pixel data - because 1366 is indivisible by 4, the number of pixels in parallel being used in the Avalon-ST Video configuration. The Avalon-ST standard handles this situation by populating the empty signal with the number of invalid (or empty) symbols in the final beat of transmission. The number of invalid symbols given in the empty signal must be a multiplication of the number of symbols per pixel in all circumstances.



2.2. Avalon-ST Video Packet Types

The three different types of Avalon-ST Video packets are defined, then a description is given of the Avalon-ST Video expected ordering and the meaning of these packets when transmitted or received by Avalon-ST Video compliant cores.

2.2.1. Avalon-ST Video Control Packets

A control packet is identified when the low nibble of the first symbol is set to decimal 15 (0xF). The Avalon-ST Video protocol further defines that any other symbol data transmitted in the first cycle (or beat) of the transmission is ignored.

An Avalon-ST Video control packet comprises the identifier nibble, and 9 other nibbles which indicate the height, width and interlacing information of any subsequent Avalon-ST Video packets.

Figure 5. Avalon-ST Video Control Packet

The figure below shows the structure of a control packet in an Avalon-ST Video configuration of two 10 bit symbols (color planes) per pixel and two pixels in parallel. Observe the symbol-alignment of the control packet nibbles; the remaining bits in each symbol are undefined.

Note: Most VIP IP cores do not drive the empty signal for control packets because extra data is always ignored. Nevertheless, a value of two indicating that the last pixel of the final beat is invalid would be tolerated.

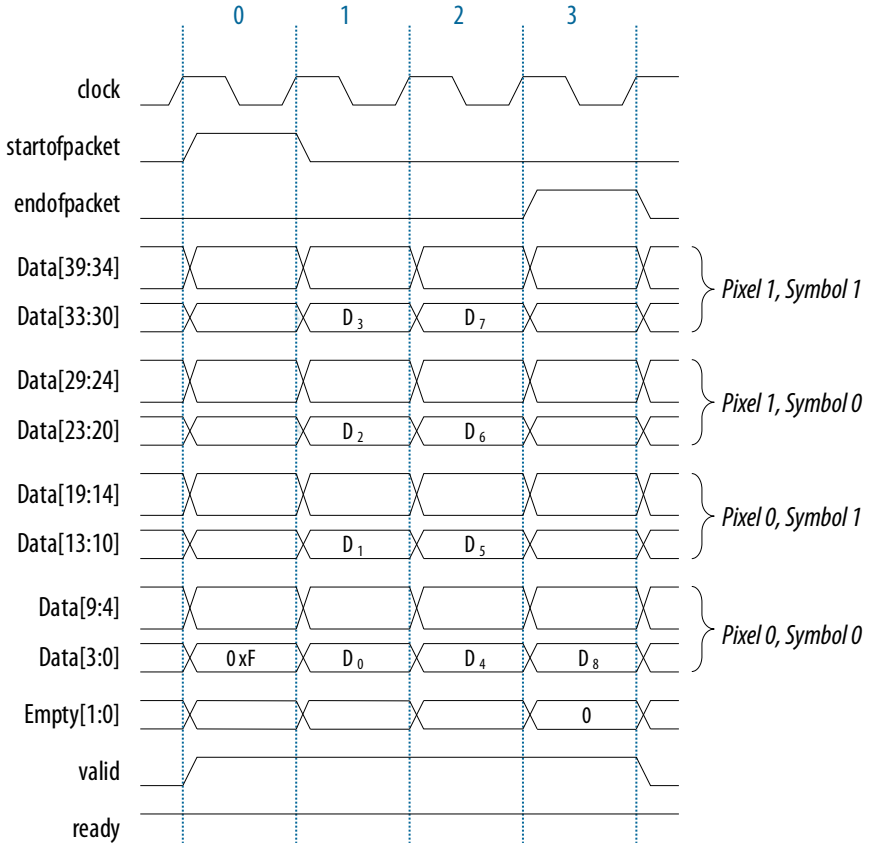


Table 7. Avalon-ST Video Control Packet Nibble Decoding

The Height and Width are given in pixels and the Interlacing nibble is decoded.

Nibble	Description
D_0	Width[15:12]
D_1	Width[11:8]

continued...

Nibble	Description
D_2	Width[7:4]
D_3	Width[3:0]
D_4	Height[15:12]
D_5	Height[11:8]
D_6	Height[7:4]
D_7	Height[3:0]
D_8	Interlacing[3:0]

When the Interlacing nibble indicates an interlaced field, the height nibble gives the height of the individual fields and not that of the equivalent whole frame—for example, a control packet for 1080i video would show a height of 540, not 1080.

Table 8. Avalon-ST Video Control Packet Interlaced Nibble Decoding

Interlaced/ Progressive	Interlacing[3]	Interlacing[2]	Interlacing[1]	Interlacing[0]	Description
Interlaced	1	1	0	0	Interlaced F1 field, paired with the following F0 field
			0	1	Interlaced F1 field, paired with the preceding F0 field
			1	x	Interlaced F1 field, pairing *don't care*
		0	0	0	Interlaced F0 field, paired with the preceding F1 field
			0	1	Interlaced F0 field, paired with the following F1 field
			1	x	Interlaced F0 field, pairing *don't care*
Progressive	0	x	0	1	Progressive frame, deinterlaced from an f1 field
			0	0	Progressive frame, deinterlaced from an f0 field
			1	x	Progressive frame

2.2.2. Avalon-ST Video Video Packets

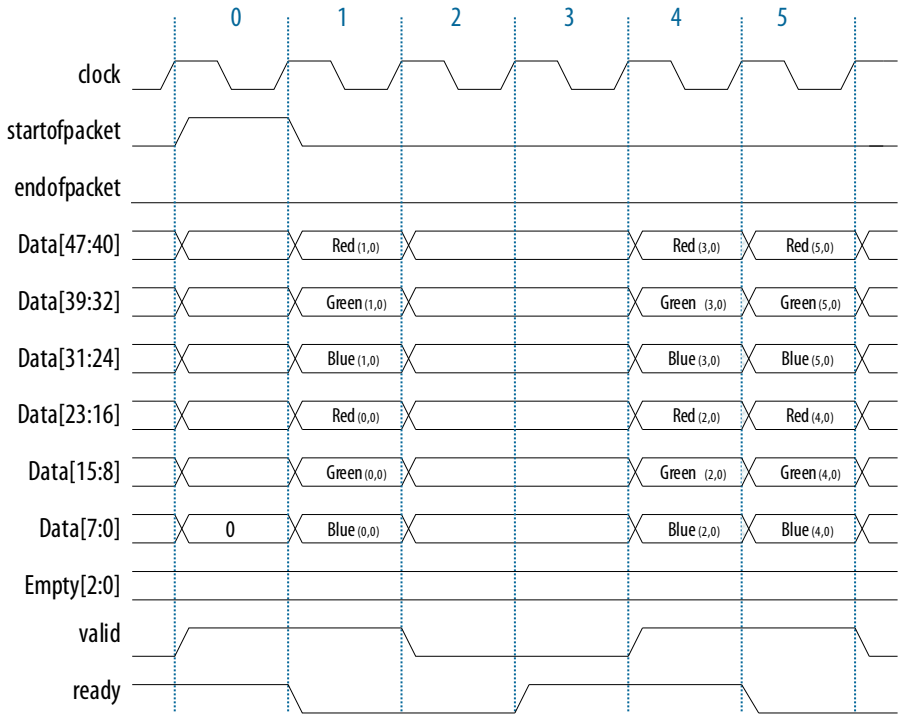
A video packet is identified when the low nibble of the first symbol is set to decimal 0.

The Avalon-ST Video protocol further defines that any other symbol data transmitted in the first cycle (or beat) of the transmission is ignored. Uncompressed and rasterized, the pixel data is transmitted in the symbols that follow in subsequent cycles, starting with the top-left pixel.

Avalon-ST Video packets support RGB and YCbCr color spaces, with 4:4:4 or 4:2:2 chroma sub-sampling, with or without an optional alpha (transparency) channel. The 4:2:0 color space is only handled by the Clocked Video interfaces and the chroma resampler. For the other VIP IP cores, the 4:2:0 color space should be converted to or from 4:2:2 for processing.

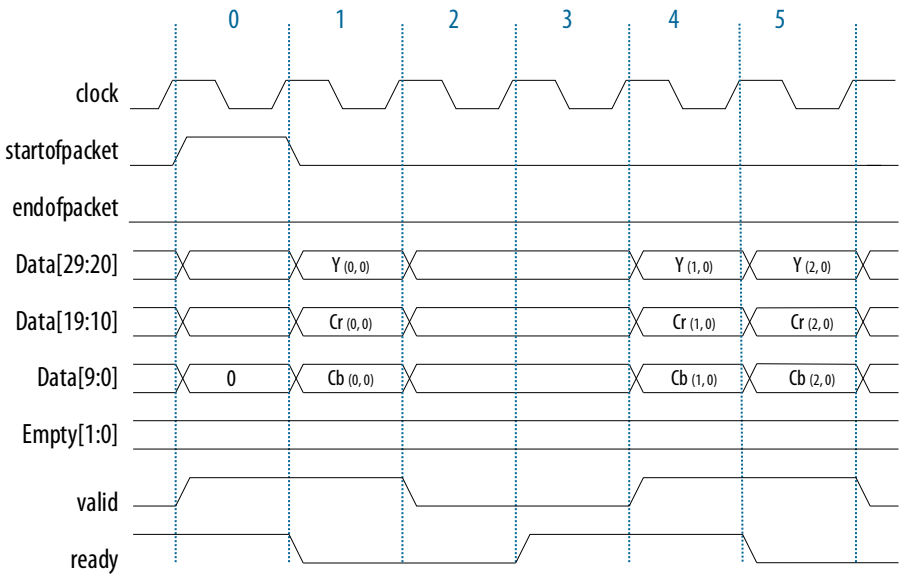
Color channel data for RGB video packets is transmitted in the order of Blue, Green, Red. You can observe this order directly on the bus for *symbols in sequence* Avalon-ST configurations. For *symbols (and pixels) in parallel* configurations, the blue symbol occupies the least significant symbol position (D_0) as shown the figure below, with the (x,y) raster position shown in brackets.

Figure 6. Avalon-ST RGB Video Packet



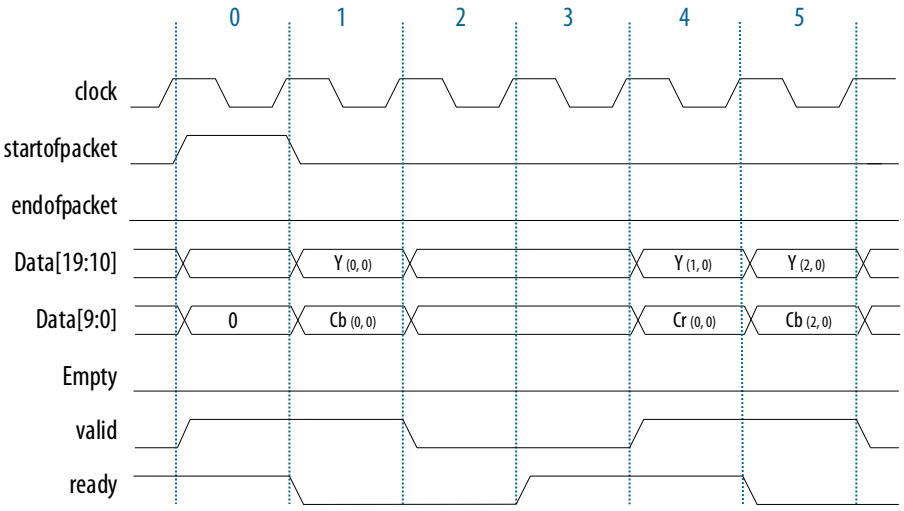
For 4:4:4 YCbCr data, chroma sample Cb is in the least significant position (or first symbol to be transmitted in a symbols in sequence configuration), Cr is the mid symbol, with the luminance (Y) occupying the most significant symbol position. The figure below shows an example with symbols in parallel.

Figure 7. Avalon-ST YCbCr 4:4:4 Video Packet



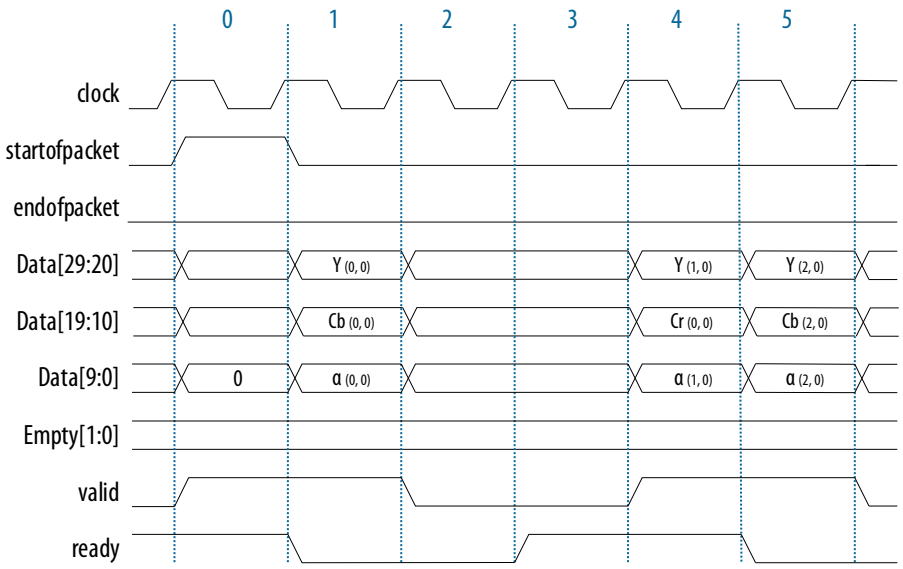
For 4:2:2 YCbCr video, with sub-sampled chroma, the Cr and Cb symbols alternate, such that each Luma symbol is associated with either a Cr or Cb symbol as shown in the figure below.

Figure 8. Avalon-ST YCbCr 4:2:2 Video Packet



For video with an alpha layer, the alpha channel occupies the first (least significant) symbol with the remaining color channels following as per the usual ordering with Blue or Cb occupying the next symbol after the alpha as shown in the figure below.

Figure 9. Avalon-ST YCbCr 4:2:2 Video Packet with Alpha Channel



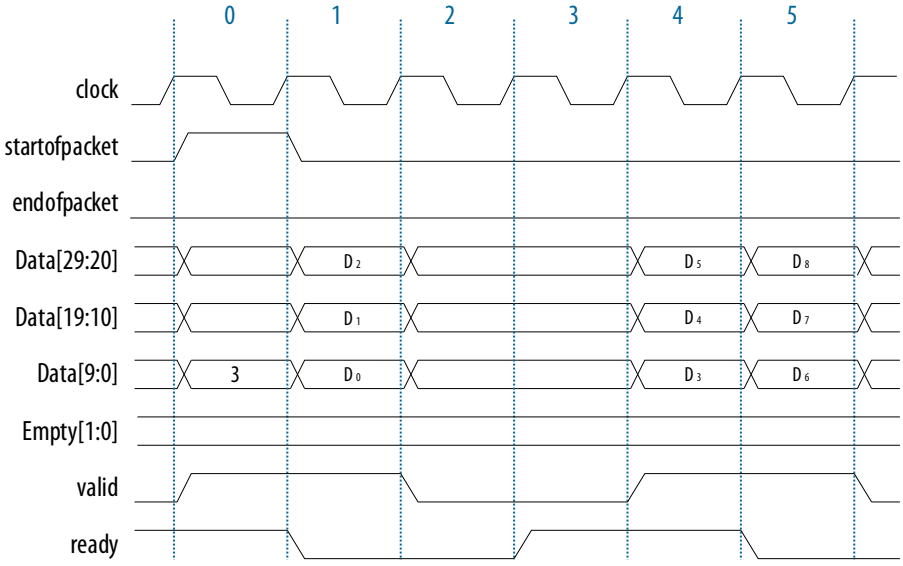
2.2.3. Avalon-ST Video User Packets

The Avalon-ST protocol may use the user packet types to transmit any user data, such as frame identification information, audio, or closed caption data.

The Avalon-ST Video protocol only requires for the payload data to begin on the second cycle of transmission (the first cycle must contain the user packet identification nibble). The content or length of these packets are ignored.

Figure 10. Avalon-ST User Packet (Start)

The figure below shows the start of an example user packet transmission. If a user packet passes through a VIP IP core which reduces the number of bits per pixel, then data is lost.



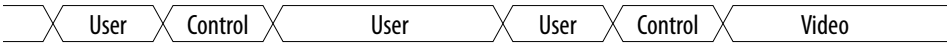
2.3. Avalon-ST Video Operation

Most Avalon-ST Video compliant VIP IP cores require an Avalon-ST control packet to be received before any video packets, so that line buffers and other sub-components can be configured.

Intel recommends that every video frame (or field, in the case of interlaced video) is preceded by a control packet. User packets may be presented in any order and may be re-ordered by some configurations of the VIP IP cores (e.g. the Deinterlacer II IP core when configured with 1 field of buffering). However, Intel recommends that the user packets precede the control packet.

Note: Some VIP IP cores, like Frame Buffer II, require a control packet to initialize storage. To ensure correct operation across all VIP components, Intel mandates that at least one control packet must be sent to an IP core prior to any video packets.

Figure 11. Avalon-ST Recommended Packet Ordering



The VIP IP cores always transmit a control packet before any video packet, and the user packets either follow or precede this control packet, depending upon the function of the IP core. When a VIP IP core receives an Avalon-ST Video control packet, the IP core decodes the height, width, and interlacing information from that packet and interprets any following Avalon-ST Video packets as being video of that format until it receives another control packet.

Most IP cores handle user packets, simply passing them through, or in the case of the Frame Buffer II IP core, writing and then reading them to memory. For IP cores that change the number of bits per symbol or symbols per pixel, additional padding is introduced to the user data.

All IP cores transmit a control packet before sending a video packet, even if no control packet has been received.

Stalling behavior (behavior when either a core is ready but there is no valid input data, or when a core has valid output data but the receiving core is not ready to receive it) varies according to the different cores. However, stalls propagate up and down the pipeline except where they can be absorbed through buffering within the cores themselves.

2.4. Avalon-ST Video Error Cases

The Avalon-ST protocol accepts certain error cases.

If a video packet has a different length to the one implied by the preceding control packet's height, width and interlacing fields, it is termed as an *early end of packet* or *late end of packet*. All the VIP IP cores are able to accept this type of erroneous video, but the resultant output video may exhibit cropped or stretched characteristics. For example, the Deinterlacer II IP core has an integral stream cleaner which allows it to accept such packets when configured in complex motion adaptive modes.

If an Avalon-ST Video packet violates the Avalon-ST protocol in some way, for example by not raising startofpacket or endofpacket, this is a more serious error case and often results in a video pipeline locking up due to a freeze of the Avalon-ST bus.

3. Clocked Video

Most IP cores in the Video and Image Processing Suite transmit and receive video according to the Avalon-ST video standard.

The Clocked Video Input II (CVI II) IP core converts clocked video into the Avalon-ST Video control and data packets and the Clocked Video Output II (CVO II) IP core converts Avalon-ST video packets into clocked video. These two IP cores interface between Avalon-ST Video cores and video interface standards such as BT.656 and others as used in Displayport, Serial Digital Interface (SDI), and High-Definition Multimedia Interface (HDMI).

Related Information

Avalon Interface Specifications
Provides more information about these interface types.

3.1. Video Formats

The Clocked Video IP cores create and accept clocked video formats.

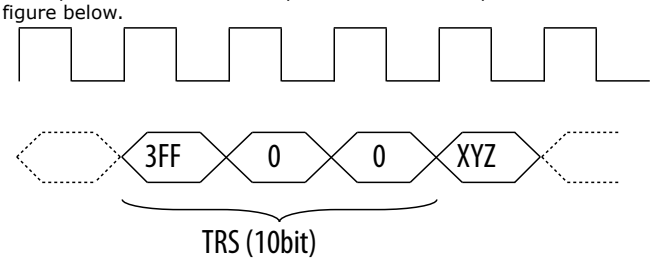
The IP cores create and accept the following formats:

- Video with synchronization information embedded in the data (in BT656 or BT1120 format)
- Video with separate synchronization (H sync, V sync) signals

The BT656 and BT1120 formats use time reference signal (TRS) codes in the video data to mark the places where synchronization information is inserted in the data.

Figure 12. Time Reference Signal Format

The TRS codes are made up of values that are not present in the video portion of the data, and they take the format shown in the figure below.



3.1.1. Embedded Synchronization Format: Clocked Video Output

For the embedded synchronization format, the CVO IP cores insert the horizontal and vertical syncs and field into the data stream during the horizontal blanking period.

The IP cores create a sample for each clock cycle on the `vid_data` bus.

There are two extra signals only used when connecting to the SDI IP core. They are `vid_trs`, which is high during the 3FF sample of the TRS, and `vid_ln`, which produces the current SDI line number. These are used by the SDI IP core to insert line numbers and cyclical redundancy checks (CRC) into the SDI stream as specified in the 1.5 Gbps HD-SDI and 3 Gbps 3G-SDI standards.

The CVO IP cores insert any ancillary packets (packets with a type of 13 or 0xD) into the output video during the vertical blanking. The IP cores begin inserting the packets on the lines specified in its parameters or mode registers (`ModeN Ancillary Line` and `ModeN F0 Ancillary Line`). The CVO IP cores stop inserting the packets at the end of the vertical blanking.

3.1.1.1. Clocked Video Output and SDI II TX Interface

The clocked video interface from the CVO II IP core expects the SDI II TX core to pull data and its attributes by asserting the CVO II IP core's read enable signal (`sdi_cvo_rden`).

Figure 13. Clocked Video Output II with SDI II TX Interface Timing Diagram

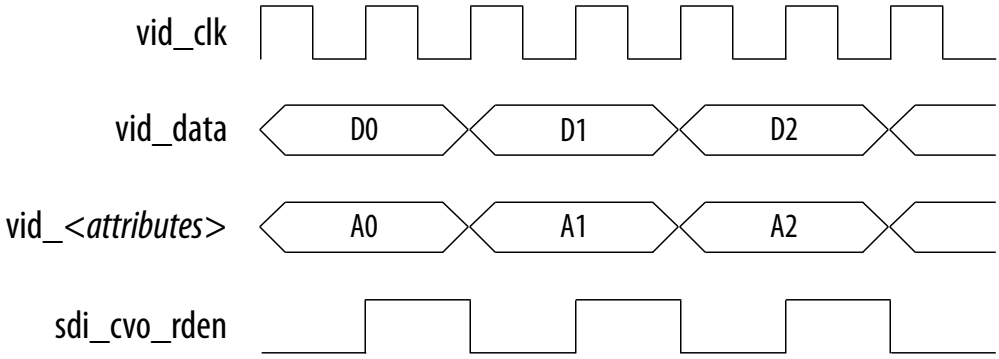
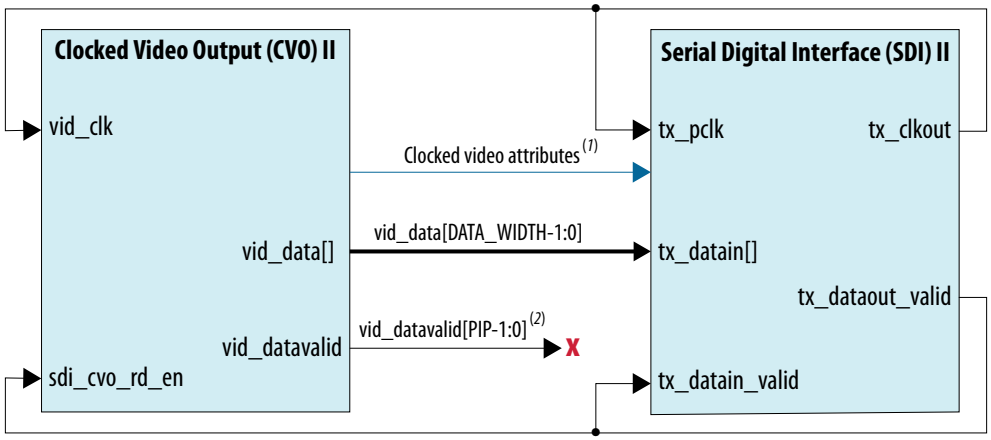


Figure 14. Clocked Video Output II with SDI II TX Interface Block Diagram



1. Clocked video attributes include signals such as vid_std, vid_trs, vid_ln, and vid_mode_change.
2. X Not used for SDI configurations.

The rate that the SDI II TX core uses to pull the data depends on the SDI standard. The table below describes the officially supported cadences.

Note: The IP core does not support SD-SDI (20 bits) mode for multi-rate designs with SDI Resampler.

Table 9. sdi_cvo_rden (tx_dataout_valid) Cadence

SDI Standard	tx_dataout_valid Cadence from SDI II TX Core
SD-SDI (10 bit)	1H 4L 1H 5L
SD-SDI (20 bit)	1H 10L
HD-SDI	1H 1L
3G-SDI	Always H
6G-SDI	Always H
12G-SDI	Always H

3.1.2. Embedded Synchronization Format: Clocked Video Input

The CVI IP cores support both 8 and 10-bit TRS and XYZ words.

When in 10-bit mode, the IP cores ignore the bottom 2 bits of the TRS and XYZ words to allow easy transition from an 8-bit system.

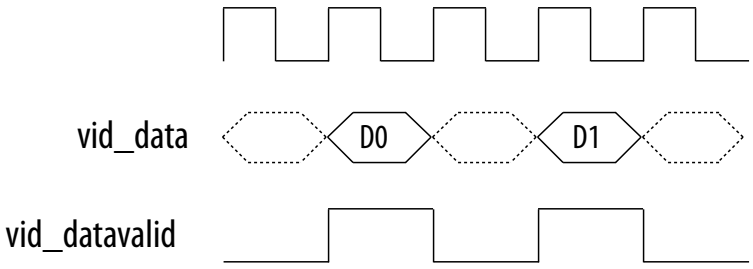
Table 10. XYZ Word Format

The XYZ word contains the synchronization information and the relevant bits of its format.

Bits	10-bit	8-bit	Description
Unused	[5:0]	[3:0]	These bits are not inspected by the CVI IP cores.
H (sync)	6	4	When 1, the video is in a horizontal blanking period.
V (sync)	7	5	When 1, the video is in a vertical blanking period.
F (field)	8	6	When 1, the video is interlaced and in field 1. When 0, the video is either progressive or interlaced and in field 0.
Unused	9	7	These bits are not inspected by the CVI IP cores.

For the embedded synchronization format, the `vid_datavalid` signal indicates a valid BT656 or BT1120 sample. The CVI IP cores only read the `vid_data` signal when `vid_datavalid` is 1.

Figure 15. Vid_datavalid Timing



The CVI IP cores extract any ancillary packets from the Y channel during the vertical blanking. Ancillary packets are not extracted from the horizontal blanking.

- Clocked Video Input IP core—The extracted packets are produced through the CVI IP core's Avalon-ST output with a packet type of 13 (0xD).
- Clocked Video Input II IP core— The extracted packets are stored in a RAM in the IP core, which can be read through the control interface.

3.1.3. Separate Synchronization Format

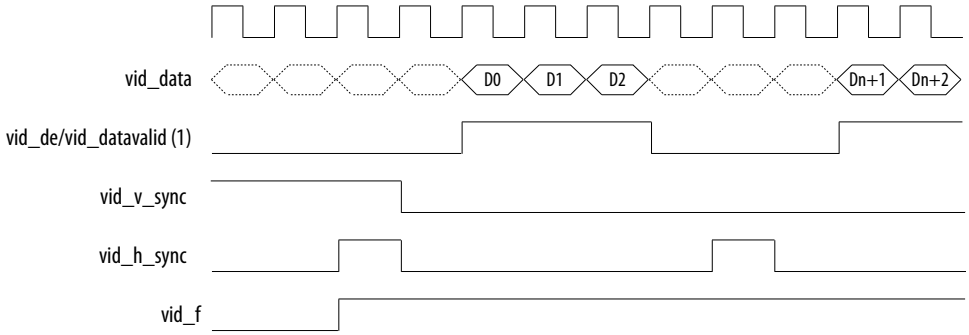
The separate synchronization format uses separate signals to indicate the blanking, sync, and field information.

The CVO IP cores create horizontal and vertical syncs and field information through their own signals. The CVO IP cores create a sample for each clock cycle on the `vid_data` bus. The `vid_datavalid` signal indicates when the `vid_data` video output is in an active picture period of the frame.

Table 11. Clocked Video Input and Output Signals for Separate Synchronization Format Video

Signal Name	Description
`vid_h_sync`	When 1, the video is in a horizontal synchronization period.
`vid_v_sync`	When 1, the video is in a vertical synchronization period.
`vid_f`	When 1, the video is interlaced and in field 1. When 0, the video is either progressive or interlaced and in field 0.
`vid_h`	When 1, the video is in a horizontal blanking period, (only for Clocked Video Output IP core).
`vid_v`	When 1, the video is in a vertical blanking period, (only for Clocked Video Output IP core).
`vid_de`	When asserted, the video is in an active picture period (not horizontal or vertical blanking). This signal must be driven for correct operation of the IP cores. *Note:* Only for Clocked Video Input IP cores.
`vid_datavalid`	• Clocked Video Output IP cores: When asserted, the video is in an active picture period (not horizontal or vertical blanking) • Clocked Video Input IP cores: Tie this signal high if you are not oversampling your input video.

Figure 16. Separate Synchronization Signals Timing Diagram



(1): vid_datavalid: Clocked Video Output IP core
vid_de: Clocked Video Input IP core

The CVI IP cores only read the `vid_data`, `vid_de`, `vid_h_sync`, `vid_v_sync`, and `vid_f` signals when `vid_datavalid` is 1. This allows the CVI IP cores to support oversampling where the video clock is running at a higher rate than the pixel clock.

3.1.4. Video Locked Signal

The `vid_locked` signal indicates that the clocked video stream is active.

When the `vid_locked` signal has a value of 1, the CVI IP cores take the input clocked video signals as valid, and read and process them as normal. When the signal has a value of 0 (if for example the video cable is disconnected or the video interface is not receiving a signal):

- Clocked Video Input IP core: The IP core takes the input clocked video signals as invalid and do not process them.
- Clocked Video Input II IP core: The `vid_clk` domain registers of the IP core are held in reset and no video is processed. The control and Avalon-ST Video interfaces are not held in reset and will respond as normal. The `vid_locked` signal is synchronized internally to the IP core and is asynchronous to the `vid_clk` signal.

If the `vid_locked` signal goes invalid while a frame of video is being processed, the CVI IP cores end the frame of video early.

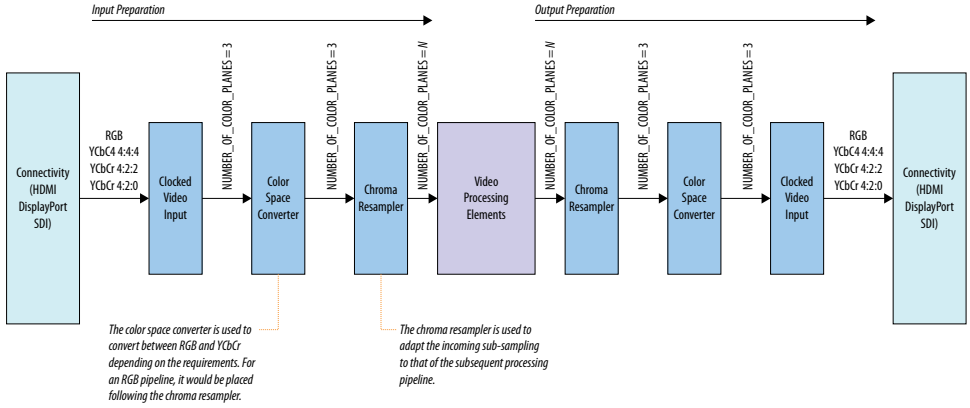
3.1.5. Clocked Video and 4:2:0 Chroma Subsampling

Other than the Chroma Resampler II IP core, none of the VIP IP cores offer explicit support for clocked video with 4:2:0 subsampling.

When processing 4:2:0 streams, you need to use chroma resampler to convert to 4:2:2 or 4:4:4 before or after any video processing is performed. The video streams can be converted back to 4:2:0 for transmission from the pipeline with another chroma resampler.

Figure 17. Pipeline with Color Space and Subsampling Adaptive Interfaces Block Diagram

The figure below shows how the chroma resampler handles the conversion.



The connectivity cores (SDI, HDMI and DisplayPort) present data at their interfaces in accordance with their respective standards, not in line with the AV-ST mappings. Before the data is processed, it must be arranged in an Avalon-ST compliant manner. The input and output preparation areas present a gray area in the pipeline where video packets are Avalon-ST video compliant but the arrangement of data within the packet may not match the expectations of the processing blocks.

3.1.5.1. Avalon-ST Video Control Packets for 4:2:0 Video

When the Chroma Resampler II IP core receives or transmits an Avalon-ST video carrying a frame in 4:2:0 format, the control packet of the associated frame will have a horizontal resolution that is half the actual resolution of the frame.

The triplet of 2 luma and 1 chroma values in 4:2:0 represent 2 pixels but they are carried in a single "pixel" of symbols in the AV-ST domain. Because the triplet is counted as a single pixel, the value of the horizontal resolution carried in control packets will be half that of the resolution the data actually represents.

For example, a UHD frame received in 4:4:4 will have a control packet specifying 3840x2160. The same resolution frame received in 4:2:0 will have a control packet indicating 1920x2160.

The chroma resampler automatically adjusts control packets depending on the conversion being applied.

- When converting from 4:2:0 to either 4:2:2 or 4:4:4, the horizontal width will be doubled.
- When converting from 4:4:4 or 4:2:2 down to 4:2:0 the horizontal width is halved.

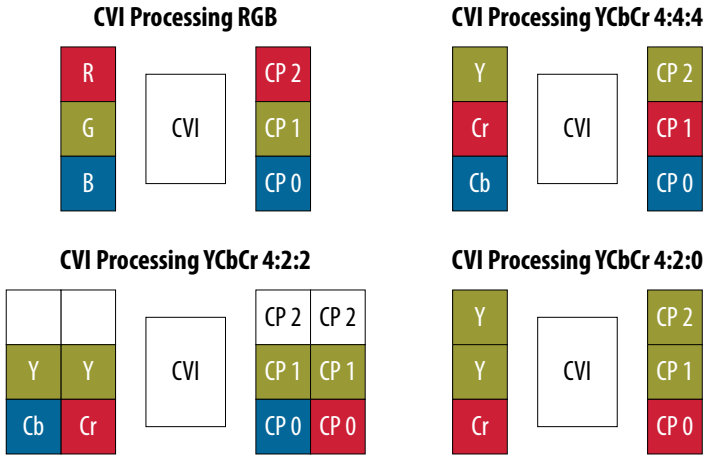
If you choose to create your own 4:2:0 processing blocks, the half horizontal width control packet requirement must be met.

3.1.5.2. 4:2:0 Clocked Video

The CVI II and CVO II IP cores are agnostic of the video standard being driven through them.

Figure 18. CVI/CVO Map Data

Figure below shows how a CVI II IP core, configured for 1 pixel in parallel with 3 color planes per pixel, maps all input pixels to the same, 3 color plane output pixel.



For 4:2:2, the "empty" data on the unused input data line becomes a color plane of "empty" data at the output of the CVI II. Likewise, the 4:2:0 triplet gets mapped into a single 3 color plane pixel at the output. This has a significant impact on handling 4:2:0.

The CVI II IP core automatically creates control packets where the horizontal width is half the real frame width. To select its timing parameters, the CVO II IP core compares the control packet dimensions against those held in the mode banks. To match a 4:2:0 control packet, the mode bank width must be recorded as half of the actual frame dimension so that it matches the control packet. If the full width is entered to the mode bank, the correct timing parameters will not be matched.

3.1.5.3. Resampling 4:2:0

When the 4:2:0 samples brought into the pipeline, they should be resampled to 4:2:2 or 4:4:4 to be compatible with the other processing IP cores.

The table below summarizes the conversions required to move from each sampling scheme to a specific pipeline sampling scheme. For systems requiring color space conversion and chroma resampling, the order of chroma resampler and color space converter in the system is determined by whether the pipeline target is RGB or YCbCr.

Table 12. Conversions Required to Present an Input Video Stream in the Designated Pipeline Format

Input Format	Pipeline Format	Conversion
RGB	RGB	None
YCbCr 4:4:4	RGB	Color Space Conversion

continued...

Input Format	Pipeline Format	Conversion
YCbCr 4:2:2	RGB	• Chroma Resampling • Color Space Conversion
YCbCr 4:2:0	RGB	• Chroma Resampling • Color Space Conversion
YCbCr 4:4:4	YCbCr 4:4:4	None
YCbCr 4:2:2	YCbCr 4:4:4	Chroma Resampling
YCbCr 4:2:0	YCbCr 4:4:4	Chroma Resampling
RGB	YCbCr 4:4:4	Color Space Conversion
YCbCr 4:4:4	YCbCr 4:2:2	Chroma Resampling
YCbCr 4:2:2	YCbCr 4:2:2	None
YCbCr 4:2:0	YCbCr 4:2:2	Chroma Resampling
RGB	YCbCr 4:2:2	• Color Space Conversion • Chroma Resampling

The Chroma Resampler II IP core makes assumptions about the arrangement of pixel sub-samples for its resampling. It could be because that the connectivity core does not supply pixels with the sub-samples in this order. If this is the case, then use a color plane sequencer to rearrange the sub-samples into the correct order.

Refer to the Chroma Resampler II IP Core on page 114 for the expected sample ordering.

4. VIP Run-Time Control

All the Video and Image Processing IP cores have an optional simple run-time control interface that comprises a set of control and status registers, accessible through an Avalon Memory-Mapped (Avalon-MM) slave port.

All the IP cores have an optional simple run-time control interface that comprises a set of control and status registers, accessible through an Avalon-MM slave port. A run-time control configuration has a mandatory set of three registers for every IP core, followed by any function-specific registers.

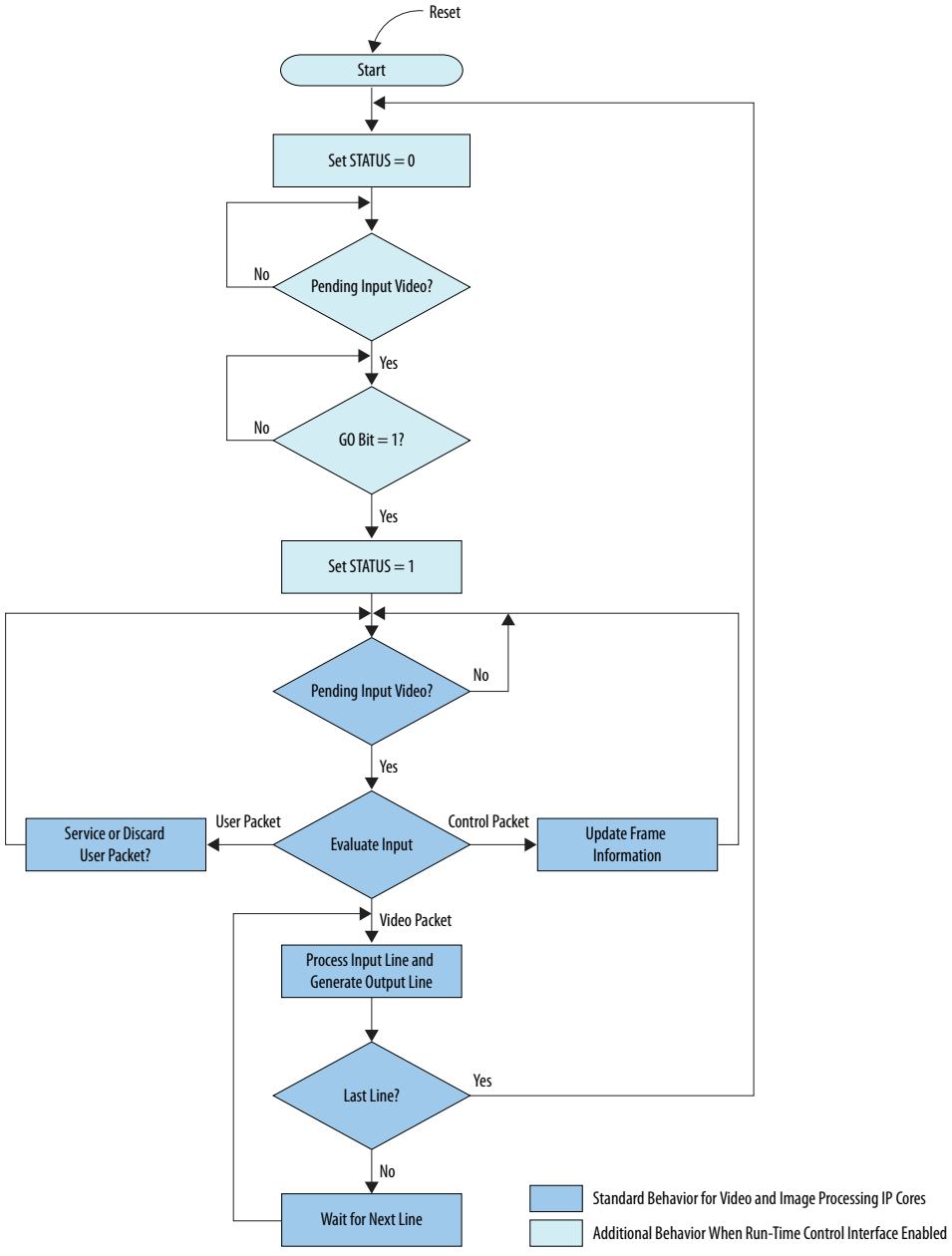
Table 13. Video and Image Processing IP Core Run-time Control Registers

Address	Data		Description
0	Bits 31:1 = X	Bit 0 = `Go`	Control register
1	Bits 31:1 = X	Bit 1 = `Status`	Status register
2	Core specific		Interrupt register

Figure 19. Video and Image Processing Suite IP Cores Behavior

The figure below illustrates the behavior of the Go and `Status` bits for every IP core when run-time control is configured, together with the steady-state running behavior that is always present.

Note: The Test Pattern Generator II and Mixer II IP cores deviate from this behavior. These IP cores start transmitting video before receiving any Avalon-ST Video packets.



When you enable run-time control, the Go bit gets deasserted by default. If you do not enable run-time control, the Go is asserted by default.

Every IP core retains address 2 in its address space to be used as an interrupt register. However this address is often unused because only some of the IP cores require interrupts.

5. Getting Started

The Video and Image Processing Suite IP cores are installed as part of the Intel Quartus Prime Standard Edition installation process.

Related Information

- Introduction to Intel FPGA IP Cores
  Provides general information about all Intel FPGA IP cores, including parameterizing, generating, upgrading, and simulating IP cores.
- Creating Version-Independent IP and Platform Designer Simulation Scripts
  Create simulation scripts that do not require manual updates for software or IP version upgrades.
- Project Management Best Practices
  Guidelines for efficient management and portability of your project and IP files.

5.1. IP Catalog and VIP Parameter Editor

The Video and Image Processing Suite IP cores are available only through the Platform Designer IP Catalog in the Intel Quartus Prime. The Platform Designer IP Catalog (**Tools ➤ Platform Designer**) and parameter editor help you easily customize and integrate IP cores into your project. You can use the Platform Designer IP Catalog and parameter editor to select, customize, and generate files representing your custom IP variation.

Double-click on any IP core name to launch the parameter editor and generate files representing your IP variation. The parameter editor prompts you to specify your IP variation name, optional ports, architecture features, and output file generation options. The parameter editor generates a top-level `.qsys` file representing the IP core in your project. Alternatively, you can define an IP variation without an open Intel Quartus Prime project. When no project is open, select the **Device Family** directly in IP Catalog to filter IP cores by device.

Use the following features to help you quickly locate and select an IP core:

- Search to locate any full or partial IP core name in IP Catalog.
- Right-click an IP core name in IP Catalog to display details about supported devices, installation location, and links to documentation.

Upgrading VIP Designs

In the Intel Quartus Prime software, if you open a design from previous versions that contains VIP components in a Platform Designer system, you may get a warning message with the title "Upgrade IP Components". This message is just letting you know that VIP components within your Platform Designer system need to be updated to their latest versions, and to do this the Platform Designer system must be regenerated before the design can be compiled within the Intel Quartus Prime

software. The recommended way of doing this with a VIP system is to close the warning message and open the design in Platform Designer so that it is easier to spot any errors or potential errors that have arisen because of the design being upgraded.

VIP IP Interoperability

The IP in the VIP Suite interoperate with each other. Their streaming interfaces use the same standard of data transmission layered on top of Avalon streaming interface. You can connect them together in Platform Designer and create a chain to build a video processing pipeline. The automatic insertion of Avalon streaming adapters in Platform Designer is disabled for video IP cores. Inserting width adapters breaks the semantics of the Avalon streaming video protocol. You must manually insert the Color Space Sequencer IP to perform data width conversion.

Related Information

Creating a System With Platform Designer
For more information on how to simulate Platform Designer designs.

5.1.1. Specifying IP Core Parameters and Options

Follow these steps to specify IP core parameters and options.

1. In the Platform Designer IP Catalog (**Tools ➤ IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name for your custom IP variation. This name identifies the IP core variation files in your project. If prompted, also specify the target FPGA device family and output file HDL preference. Click **OK**.
3. Specify parameters and options for your IP variation:
   - Optionally select preset parameter values. Presets specify all initial parameter values for specific applications (where provided).
   - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
   - Specify options for generation of a timing netlist, simulation model, testbench, or example design (where applicable).
   - Specify options for processing the IP core files in other EDA tools.

   Note: The Video and Image Processing IP cores support only ModelSim* simulation software.
4. Click **Finish** to generate synthesis and other optional files matching your IP variation specifications. The parameter editor generates the top-level `.qsys` IP variation file and HDL files for synthesis and simulation. Some IP cores also simultaneously generate a testbench or example design for hardware testing.
5. To generate a simulation testbench, click **Generate ➤ Generate Testbench System**. **Generate Testbench System** is not available for some IP cores that do not provide a simulation testbench.
6. To generate a top-level HDL example for hardware verification, click **Generate ➤ HDL Example**. **Generate ➤ HDL Example** is not available for some IP cores.

The top-level IP variation is added to the current Intel Quartus Prime project. Click **Project ➤ Add/Remove Files in Project** to manually add a `.qsys` (Intel Quartus Prime Standard Edition) or `.ip` (Intel Quartus Prime Pro Edition) file to a project. Make appropriate pin assignments to connect ports.

5.2. Installing and Licensing IP Cores

The Intel Quartus Prime software installation includes the Intel FPGA IP library. This library provides useful IP core functions for your production use without the need for an additional license. Some Intel FPGA IP IP functions in the library require that you purchase a separate license for production use. The OpenCore® feature allows evaluation of any Intel FPGA IP core in simulation and compilation in the Intel Quartus Prime software. Upon satisfaction with functionality and performance, visit the Self Service Licensing Center to obtain a license number for any Intel FPGA product.

The Intel Quartus Prime software installs IP cores in the following locations by default:

Figure 20. IP Core Installation Path

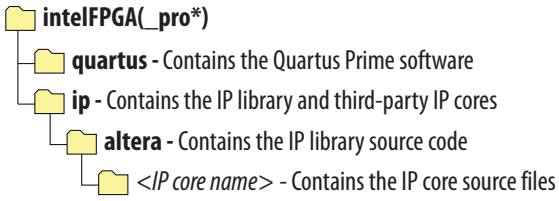


Table 14. IP Core Installation Locations

Location	Software	Platform
<*drive*>:\intelFPGA_pro\quartus\ip\altera	Intel Quartus Prime Pro Edition	Windows
<*drive*>:\intelFPGA\quartus\ip\altera	Intel Quartus Prime Standard Edition	Windows
<*home directory*>:/intelFPGA_pro/quartus/ip/altera	Intel Quartus Prime Pro Edition	Linux*
<*home directory*>:/intelFPGA/quartus/ip/altera	Intel Quartus Prime Standard Edition	Linux

5.2.1. Intel FPGA IP Evaluation Mode

The free Intel FPGA IP Evaluation Mode allows you to evaluate licensed Intel FPGA IP cores in simulation and hardware before purchase. Intel FPGA IP Evaluation Mode supports the following evaluations without additional license:

- Simulate the behavior of a licensed Intel FPGA IP core in your system.
- Verify the functionality, size, and speed of the IP core quickly and easily.
- Generate time-limited device programming files for designs that include IP cores.
- Program a device with your IP core and verify your design in hardware.

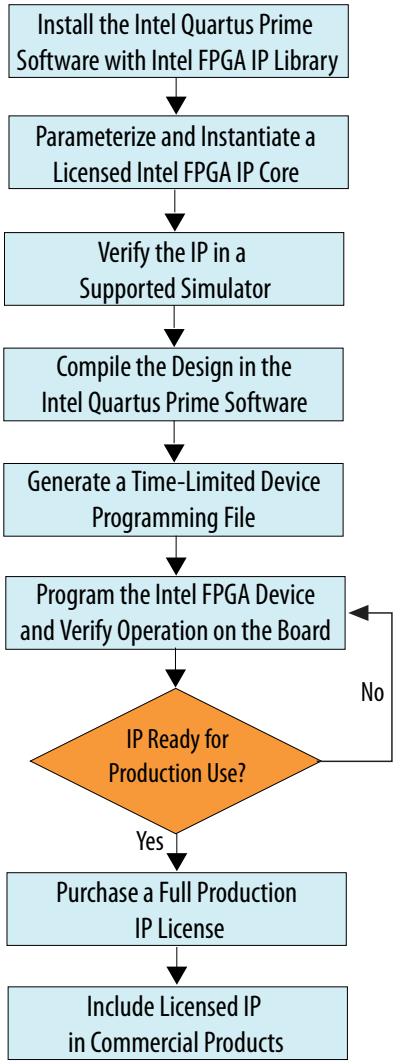
Intel FPGA IP Evaluation Mode supports the following operation modes:

- **Tethered**—Allows running the design containing the licensed Intel FPGA IP indefinitely with a connection between your board and the host computer. Tethered mode requires a serial joint test action group (JTAG) cable connected between the JTAG port on your board and the host computer, which is running the Intel Quartus Prime Programmer for the duration of the hardware evaluation period. The Programmer only requires a minimum installation of the Intel Quartus Prime software, and requires no Intel Quartus Prime license. The host computer controls the evaluation time by sending a periodic signal to the device via the JTAG port. If all licensed IP cores in the design support tethered mode, the evaluation time runs until any IP core evaluation expires. If all of the IP cores support unlimited evaluation time, the device does not time-out.
- **Untethered**—Allows running the design containing the licensed IP for a limited time. The IP core reverts to untethered mode if the device disconnects from the host computer running the Intel Quartus Prime software. The IP core also reverts to untethered mode if any other licensed IP core in the design does not support tethered mode.

When the evaluation time expires for any licensed Intel FPGA IP in the design, the design stops functioning. All IP cores that use the Intel FPGA IP Evaluation Mode time out simultaneously when any IP core in the design times out. When the evaluation time expires, you must reprogram the FPGA device before continuing hardware verification. To extend use of the IP core for production, purchase a full production license for the IP core.

You must purchase the license and generate a full production license key before you can generate an unrestricted device programming file. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (<*project name*>`_time_limited.sof`) that expires at the time limit.

Figure 21. Intel FPGA IP Evaluation Mode Flow



Note: Refer to each IP core's user guide for parameterization steps and implementation details.

Intel licenses IP cores on a per-seat, perpetual basis. The license fee includes first-year maintenance and support. You must renew the maintenance contract to receive updates, bug fixes, and technical support beyond the first year. You must purchase a full production license for Intel FPGA IP cores that require a production license, before generating programming files that you may use for an unlimited time. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (`<project name>_time_limited.sof`) that expires at the time limit. To obtain your production license keys, visit the Intel FPGA Self-Service Licensing Center.

The Intel FPGA Software License Agreements govern the installation and use of licensed IP cores, the Intel Quartus Prime design software, and all unlicensed IP cores.

Related Information

- Intel FPGA Licensing Support Center
- Introduction to Intel FPGA Software Installation and Licensing

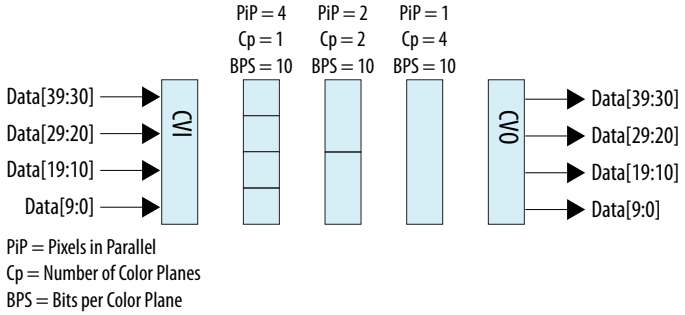
6. VIP Connectivity Interfacing

Avalon-ST Video expects pixel subsamples to be arranged in particular orders, depending on the sampling method selected.

While the **Color planes transmitted in parallel** and **Number of color planes** parameters define an interface that is capable of carrying the sampling methods, they do not enforce the transmission of particular sub-samples in particular symbols of the Avalon-ST video packet.

You have to understand the arrangement of the color planes on entry to the pipeline and any reconfiguration of this order performed by the components within the pipeline. This is a particular concern around the connectivity points. The connectivity IP cores present data arranged according to their respective standards. When connected to a clocked video component, the clocked video components will package the data as it is presented to the IP core. They do not re-arrange it. In simple terms, on each clock cycle during the active video, the Clocked Video Input (CVI) II IP samples the entire data bus and divides the samples into pixels according to the **Number of color planes**, **Bits per color plane**, and **Pixels in parallel** parameters used to configure the module.

Figure 22. Variable Interpretation of Pixels on a Clocked Video Data Bus Based on Parameterization



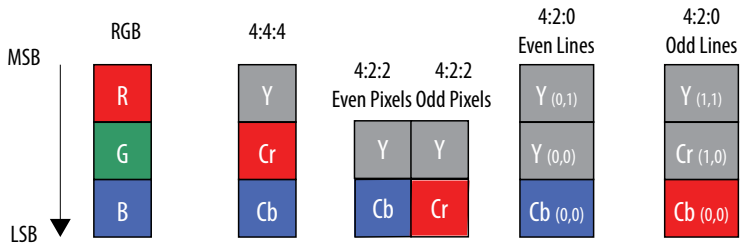
If the configuration selected were PiP=1 and CP=4, but a 10-bit RGB signal were being fed on Data [29:30], the output pixels will still be 40 bits in size, the unused data bits having been sampled.

The converse function of the Clocked Video Output (CVO) II IP drives the entire data bus on each clock cycle. To drive 10-bit RGB on Data[29:0] in the PiP=1 and CP=4 configuration, the VIP pipeline would have to generate 40-bit pixels containing the 30 data bits and 10 null bits.

6.1. Avalon-ST Color Space Mappings

The Avalon-ST expected arrangement differs for the various supported color spaces.

Figure 23. Expected Order of Chroma Samples for Avalon-ST Video

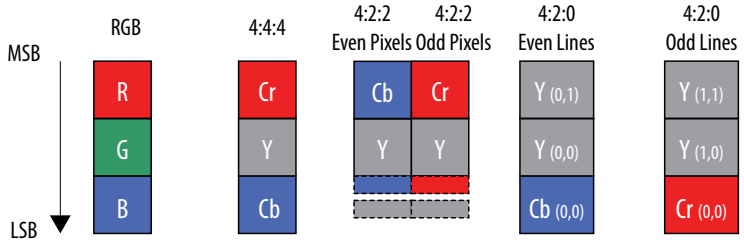


The CVI and CVO blocks offer a simple mechanism to present data in the AV-ST video format but they do not offer functions to remap the many different data mappings the connectivity cores present.

6.1.1. Interfacing with High-Definition Multimedia Interface (HDMI)

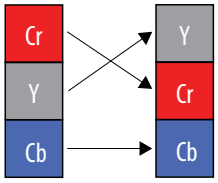
The order that the HDMI core presents chroma samples differs from the Avalon-ST expectations for YCbCr 4:4:4 and 4:2:2 sampling schemes.

Figure 24. Intel HDMI IP Core Chroma Sampling



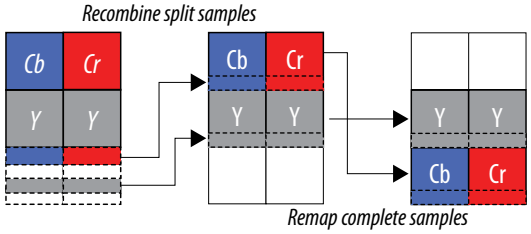
For YCbCr 4:4:4, it is necessary to perform the translation shown in the figure below to meet the Avalon-ST requirements. If the system only handles YCbCr, then you can use the Color Plane Sequencer II IP core to perform this remapping. If the system handles both RGB and YCbCr, then you need to use the Color Space Converter II IP core to convert between RGB and YCbCr and also to remap the color plane ordering for YCbCr 4:4:4.

Figure 25. Remapping HDMI 4:4:4 to Avalon-ST Video 4:4:4



YCbCr 4:2:2 require two potential remappings. Symbols 2 and 1 carry the upper 8 bits of the chroma and luma samples. If the system supports 10- or 12-bit depth, the additional bits are carried together in symbol 0. To support 10 or 12 bit, YCbCr 4:2:2 requires the recombining of these lower bits with the upper bits in 2 distinct symbols as shown in the figure below.

Figure 26. Remapping HDMI YCbCr 4:2:2 to Avalon-ST YCbCr 4:2:2



At present, a Mux instantiated between the HDMI IP core and the clocked video IP core implements the recombining of the 4:2:2 MSBs with LSBs. Future clocked video IP cores will support this remapping internally.

Note: For 8-bit inputs, recombination is not necessary.

The positioning of the luma and chroma in the upper 2 symbols is at odds with the Avalon-ST requirement for the chroma in the bottom symbol and luma in the symbol above. The luma samples are in the correct place but the chroma samples must be remapped from the upper symbol to lower symbol.

If the system only handles YCbCr, then you can use the Color Plane Sequencer II IP core to remap the symbols. If the system handles both RGB and YCbCr, then you need to use the Color Space Converter II IP core to remap the chroma samples.

6.1.2. Interfacing with DisplayPort

YCbCr 4:4:4 requires the same translation as the HDMI IP core.

Figure 27. Remapping DisplayPort 4:4:4 to Avalon-ST Video 4:4:4

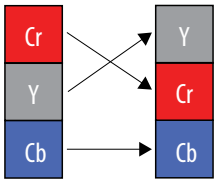
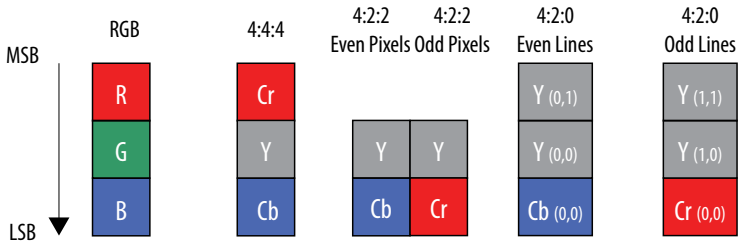


Figure 28. Intel DisplayPort IP Core Chroma Sampling



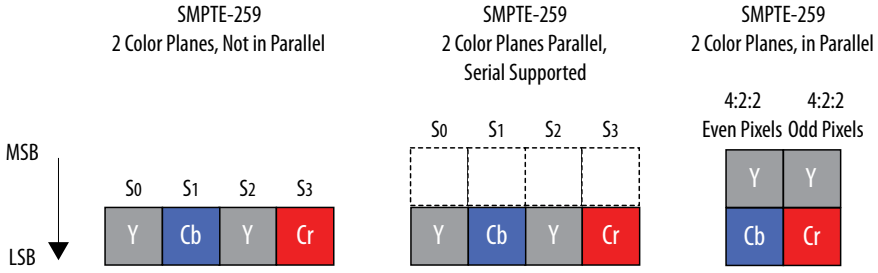
If the system only handles YCbCr, then you can use the Color Plane Sequencer II IP core to perform the remapping. If the system handles both RGB and YCbCr, then you need to use the Color Space Converter II IP core to convert between RGB and YCbCr and also to remap the color plane ordering for YCbCr 4:4:4.

6.1.3. Interfacing with Serial Digital Interface (SDI)

The succession of SDI standards offers a range of mappings of video samples onto SDI symbols.

The SDI-clocked video interface deals in the SDI symbols, not the video samples. This means that only certain of the SDI mappings are directly compatible with Avalon-ST video requirements. The figure below shows the 8- and 10-bit YCbCr 4:2:2 mappings.

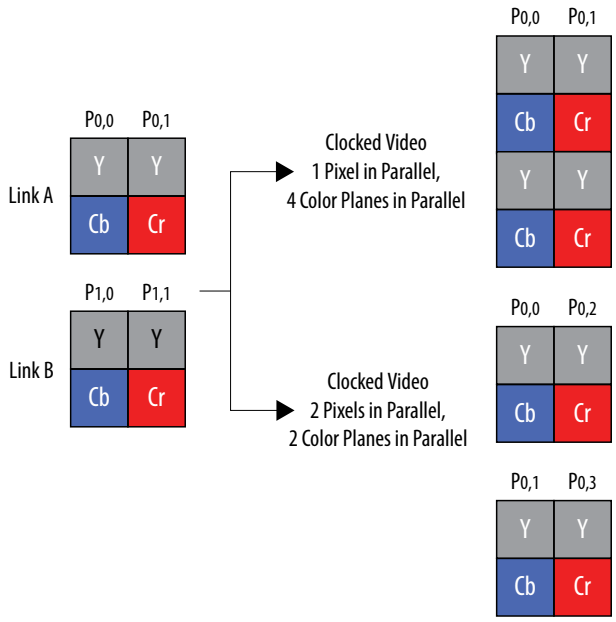
Figure 29. The SDI Sample Mappings Directly Compatible with Avalon-ST Video



You can transport other mappings into the Avalon-ST domain but to rearrange the samples into a compatible format, immediate remapping into the Avalon-ST format is required. The clocked video units do not perform any remapping function.

Figure 30. SMPTE-372 Dual Link Carrying Dual Streams

YCbCr 4:2:2 delivered as SMPTE-372 dual-link format over 2 20 bit buses can be configured either as a single pixel of 4 parallel color planes or 2 pixels of 2 parallel color planes.

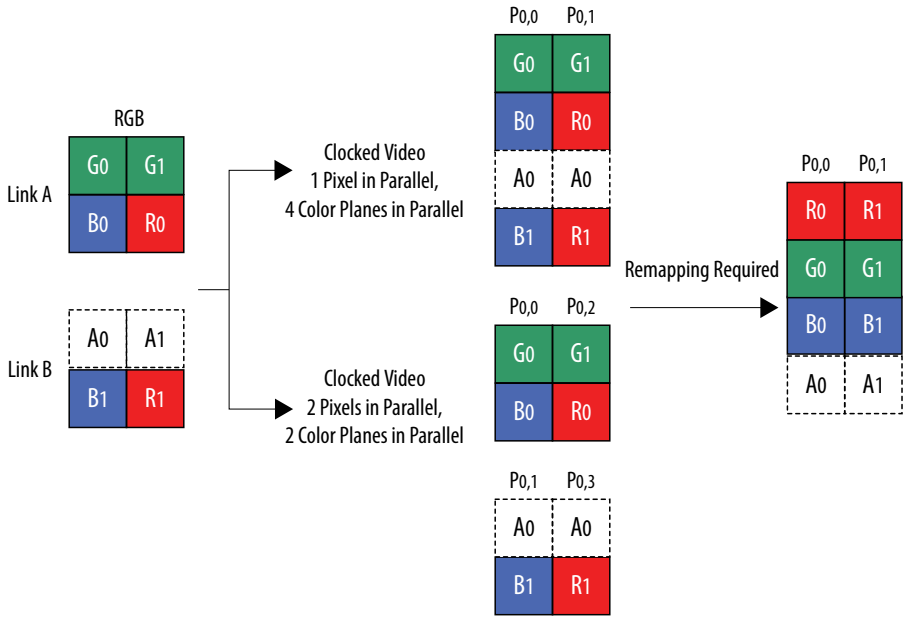


If a single pixel configuration is used, then the horizontal resolution would be correct in the Avalon-ST control packet. If the 2 pixel solution were selected, the Avalon-ST control packet would report a width 2x the actual horizontal resolution (since an entire extra line is carried on Link B but it's pixels would be counted as part of Link As). In

both cases, the Avalon-ST control packet would report a vertical resolution ½ the actual resolution since 2 lines have been handled as 1. Any remapping logic has to account for this.

Figure 31. Clocked Video Input Interpretations of SMPTE-372 Dual Link 10-bit RGBA Mapping

For 10-bit RGBA carried on a SMPTE-372 dual link remapping is required since 2 cycles of samples are required to assemble complete RGB pixel.



In this case using 1 pixel, 4 color planes in parallel will mean that the control packet reflects the true dimensions of the incoming image. If 2 pixels with 2 color planes is configured, then the control packet will report a horizontal width 2x the actual width.

The 12-bit data mappings also require processing to recombine the relevant SDI symbols into the correct pixel components.

Figure 32. Clocked Video Input Interpretations of SMPTE-372 Dual Link 12-bit RGBA Mapping

The12-bit RGB mapping matches the 12-bit YCbCr 4:4:4 mapping if you replace R with Cr, G with Y and B with Cb.

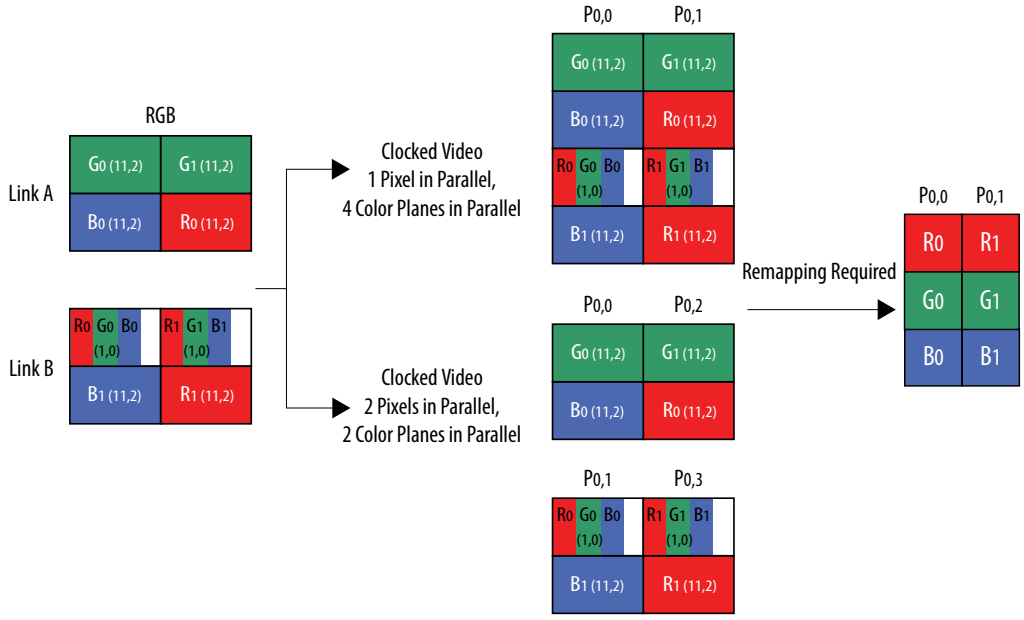
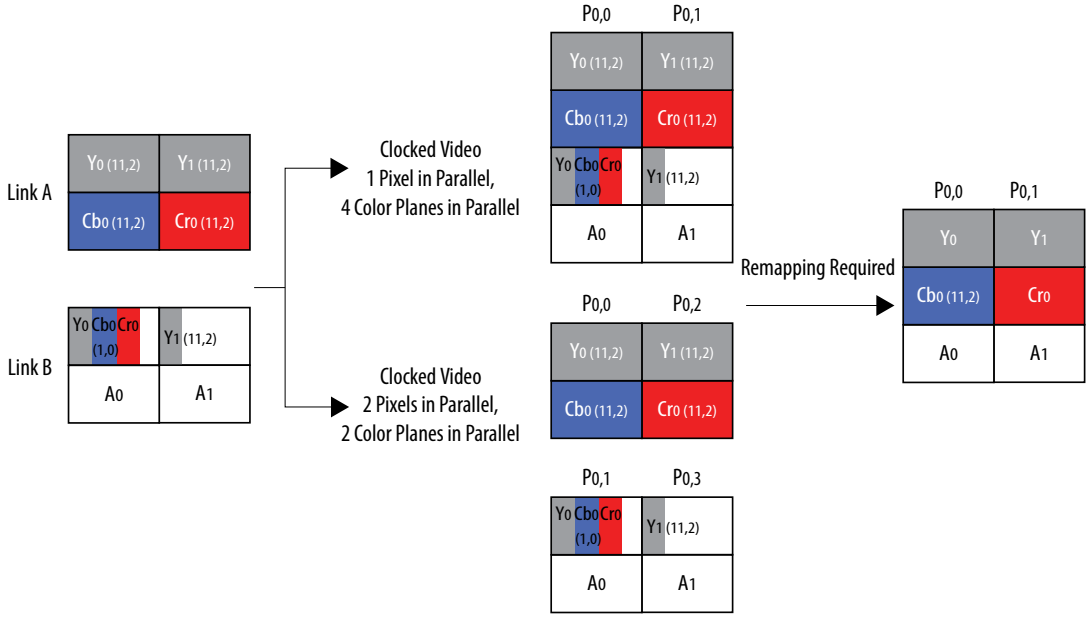


Figure 33. Clocked Video Input Interpretations of SMPTE-372 Dual Link 12-bit YCbCr Mapping

The 12 bit 4:2:2 mapping with optional alpha channel also spreads pixel components across multiple SDI symbols requiring remapping to an Avalon-ST Video compliant arrangement.



If you select 2 pixel in parallel configuration, then the control packets from a CVI will report 2x actual width. Going into the CVO, the packets would need to report 2x the actual width.

6.1.4. Unsupported SDI Mappings

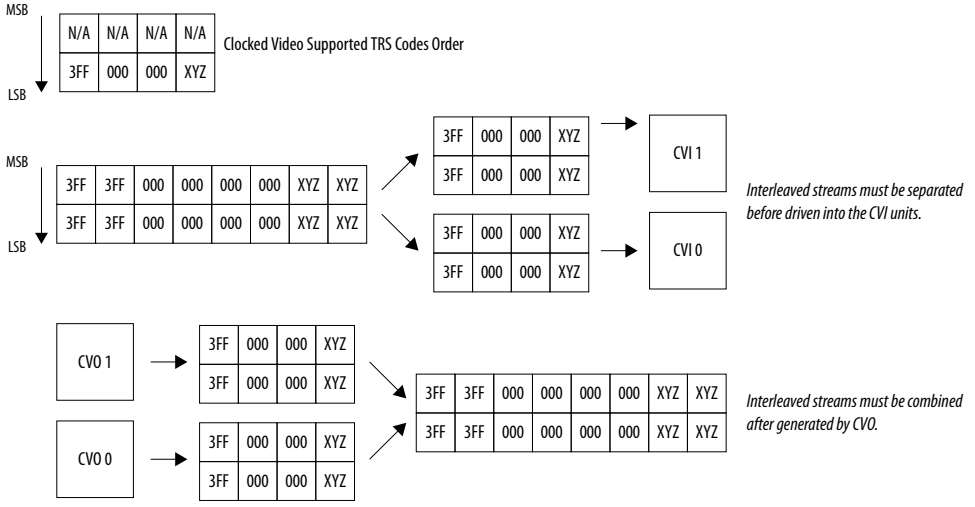
The key requirement for compatibility with the clocked video units is that the TRS code words be presented in the 10 LSBs of the clocked video interface in the order of 3FF, 000, 000, XYZ.

The CVI can only identify streams demarked in this way. The CVO can only generate streams demarked in this way.

This means that the SMPTE-425 Level B mappings are not directly supported by the clocked video units. Because both Level B-DL and Level B-DS mappings modify the arrival order of TRS codes, these streams must be demultiplexed before clocked video inputs. After clocked video outputs, the streams must be multiplexed together.

Figure 34. Clocked Video Requirements of TRS Codes

The figure below shows that the clocked video units do not handle dual link and dual stream mappings in a single stream (SMPTE-425 level B). Separation of streams must be handled before entry to the CVI. Dual CVO outputs must be recombined into a single stream after the CVO.



6.1.5. 12G-SDI

The CVI II and CVO II IP cores support two-sample interleave (2SI) format variant of the 12G-SDI specification. The IP cores do not support square divisions quad split format.

You can enable the support for 12G-SDI by turning on the **Support 6G and 12G-SDI** parameter in the CVI II and CVO II IP parameter editors. Turning on the **Support 6G and 12G-SDI** parameter enables the SDI Resampler block within a CVI II or CVO II instance. The SDI resampler enables the support for 12G-SDI format.

The SDI Resampler also supports 6G-, 3G-, HD-, and SD-SDI formats. Conversion to Avalon-ST video format is possible only for configurations with 4 pixels in parallel. With the presence of the SDI Resampler, the CVI II and CVO II IPs are able to provide seamless switching between 12G-, 6G-, 3G-, HD-, and SD-SDI formats.

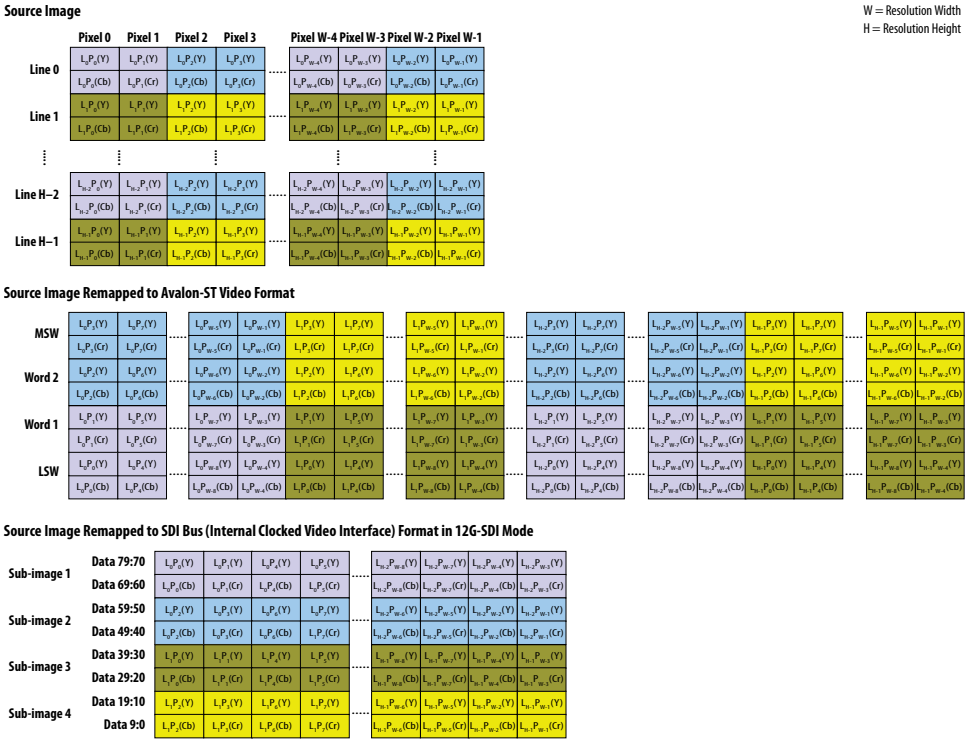
Note: The **Support 6G and 12G-SDI** parameter is available only for embedded sync modes..

6.1.5.1. 12G-SDI with Clocked Video Input II

You can configure the CVI II IP core to allow 12G-SDI packets pass into the Avalon-ST video domain.

The 12G-SDI option uses the two-sample interleave (2SI) pattern, which transmits two lines simultaneously, The simultaneous transmission of two lines means that the active width received by the CVI II IP core in between two sets of horizontal SYNC is two times the actual width. The CVI II IP core applies the appropriate scaling before reporting to the `Active Sample Count` status register and the Avalon-ST video control packet. The 2SI pattern also requires that the clocked video input to be remapped to raster scan order for processing in a VIP pipeline. The CVI II IP core performs the remapping.

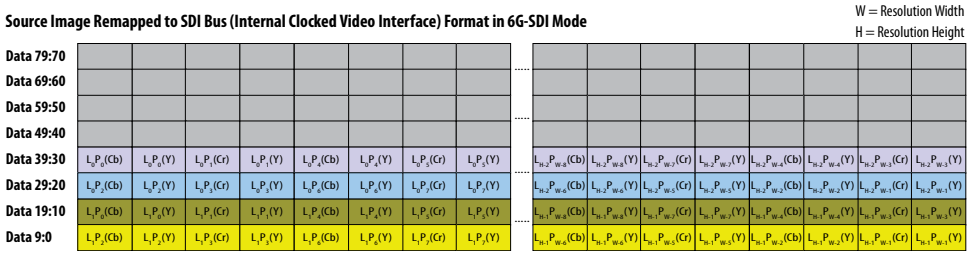
Figure 35. Clocked Video Capture of 12G-SDI Mapping and Resampling (Remapping) for Avalon-ST Video



When a 6G-SDI packet is carried over the 12G link, half of the 80-bit data bus will be unused. The SDI Resampler within the CVI II IP core discards the null pixels, splits the pixel components, and accumulates successive components to recombine the original image format into Avalon-ST format.

Note: The SDI Resampler is only enabled when you turn on the **Support 6G and 12G-SDI** parameter with the pixels in parallel fixed to 4 in the CVI II or CVO II Intel FPGA IP parameter editor.

Figure 36. Clocked Video Capture of 6G-SDI Mapping Transported Over 12G-SDI Link and Required Remapping



When a 3G-SDI packet is carried over a 12G-SDI link, the CVI II IP core captures 3 null pixels for every active pixel. The SDI Resampler discards these null pixels to reestablish the 3G-SDI sample order in the Avalon-ST video frame, as shown in the figure below.

Figure 37. Clocked Video Capture of 3G-SDI Mapping Transported Over 12G-SDI Link

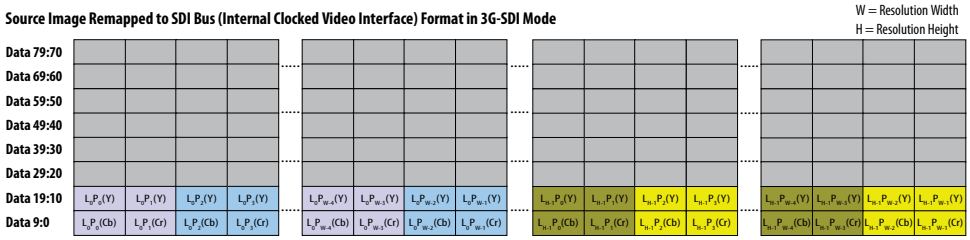


Table 15. SDI Resampler with CVI II IP Parameters

The table shows the differences in the CVI II IP core with the presence of the SDI Resampler.

Parameters	Support 6G and 12G-SDI Off	Support 6G and 12G-SDI On
SDI Resampler	Not available	Present
Pixels in parallel	1	4
Standard width range	1–16	3
Multi-rate Mode	3G-SDI, HD-SDI, SD-SDI	12G-SDI, 6G-SDI, 3G-SDI, HD-SDI, SD-SDI
Square Division Quad format	Not supported	Not supported
2 Sample-interleave (2SI) format	Not supported	Supported
12G-SDI with 8 streams interleaved	Not supported	Supported
12G-SDI with 16 streams interleaved	Not supported	Not supported
6G-SDI with 4 streams interleaved	Not supported	Not supported
6G-SDI with 8 streams interleaved	Not supported	Supported
3G-SDI Level A	Supported	Supported
3G-SDI Level B	Not supported	Not supported
HD-SDI	Supported	Supported
SD-SDI	Supported	Supported
4:4:4	Not supported	Not supported

continued...

Parameters	Support 6G and 12G-SDI Off	Support 6G and 12G-SDI On
4:2:2	Supported	Supported
4:2:0	Not supported	Not supported
Minimum active frame size	32x32	• 3G-SDI, HD-SDI, SD-SD: 32x32 • 12G-SDI, 6G-SDI: 32x64
Active width supported	Modulo 1	• 12G-SDI, 6G-SDI, 3G-SDI, HD-SDI: Modulo 4 • SD-SD: Modulo 8
Active height supported	Modulo 1	Modulo 1
Clipping	Supported	Not supported
Padding	Supported	Not supported

6.1.5.2. 12G-SDI with Clocked Video Output II

The mapping and reordering of the pixels for clocked video outputs are the reverse of the clocked video inputs.

Table 16. SDI Resampler with CVO II IP Parameters

The table shows the differences in the CVO II IP core with the presence of the SDI Resampler.

Parameters	Support 6G and 12G-SDI Off	Support 6G and 12G-SDI On
SDI Resampler	Not available	Present
Pixels in parallel	1	4
Standard width range	1–16	3
Multi-rate Mode	3G-SDI, HD-SDI, SD-SDI	12G-SDI, 6G-SDI, 3G-SDI, HD-SDI, SD-SDI
Square Division Quad format	Not supported	Not supported
2 Sample-interleave (2SI) format	Not supported	Supported
12G-SDI with 8 streams interleaved	Not supported	Supported
12G-SDI with 16 streams interleaved	Not supported	Not supported
6G-SDI with 4 streams interleaved	Not supported	Not supported
6G-SDI with 8 streams interleaved	Not supported	Supported
3G-SDI Level A	Supported	Supported
3G-SDI Level B	Not supported	Not supported
HD-SDI	Supported	Supported
SD-SDI	Supported	Supported
4:4:4	Not supported	Not supported
4:2:2	Supported	Supported
4:2:0	Not supported	Not supported
Minimum active frame size	32x32	Fixed resolution: 32x32

continued...

Parameters	Support 6G and 12G-SDI Off	Support 6G and 12G-SDI On
		Note: For switching resolutions, if the frame is smaller than the combined capacity of the CVO IP core's FIFOs and pipelines, the behavior is undetermined. Use the set of standard resolutions defined in the CTA-861-G specifications.
Active width supported	Modulo 1	• 12G-SDI, 6G-SDI, 3G-SDI, HD-SDI: Modulo 4 • SD-SDI: Modulo 8
Active height supported	Modulo 1	• 12G-SDI, 6G-SDI: Modulo 2 • 3G-SDI, HD-SDI, SD-SD: Modulo 1
Dynamic control over Go bit	Supported	Cannot be disabled once enabled.
Low latency mode	Supported	Not supported

Note: In simulation, you can create artificial short bursts of just a few frames before switching either video standard or video resolution, or both. If two back-to-back switches occur very rapidly e.g. switching from A-to-B-to-C or A-to-B-to-A, the clocked video output's behavior is nondeterministic.

7. Clocked Video Interface IPs

The Clocked Video Interface IPs convert clocked video formats (such as BT656, BT1120, and DVI) to Avalon streaming video; and vice versa. You can configure these IP at run time using an Avalon memory-mapped target interface.

The Clocked Video Input II IP:

- Converts clocked video formats (such as BT656, BT1120, and DVI) to Avalon streaming video.
- Provides clock crossing capabilities to allow video formats running at different frequencies to enter the system.
- Strips incoming clocked video of horizontal and vertical blanking, leaving only active picture data.

The Clocked Video Output II IP:

- Converts data from the flow controlled Avalon streaming video protocol to clocked video.
- Formats Avalon streaming video into clocked video. Inserts horizontal and vertical blanking and generates horizontal and vertical synchronization information using the Avalon streaming video control and active picture packets.
- Provides clock crossing capabilities to allow video formats running at different frequencies to be created from the system.

7.1. Supported Features for Clocked Video Output II IP

The Clocked Video Output II IP support the following features.

- HDMI SD/HD
- HDMI 4K
- 3G-SDI mode 1
- 12G-SDI

  Note: The IP does not support Low Latency Mode if you turn on the **Use 6G and 12G-SDI** parameter.
- Low latency mode

  Note: The IP does not support Low Latency Mode if you turn on the **Use 6G and 12G-SDI** parameter.
- Full frame mode

7.2. Control Port

To configure a clocked video IP using an Avalon memory-mapped target interface, turn on **Use control port** in the parameter editor.

Initially, the IP is disabled and does not transmit any data or video. However, the Clocked Video Input IP still detects the format of the clocked video input and raise interrupts; and the Clocked Video Output IP still accepts data on the Avalon streaming video interface if the input FIFO buffer has space.

The sequence for starting the output of the IP:

1. Write a 1 to `Control` register bit 0.
2. Read `Status` register bit 0. When this bit is 1, the IP starts transmitting data or video. The transmission starts on the next start of frame or field boundary.

   Note: For the Clocked Video Input IP, the frame or field matches the **Field order** parameter settings.

The sequence for stopping the output of the IP:

1. Write a 0 to `Control` register bit 0.
2. Read `Status` register bit 0. When this bit is 0, the IP stops transmitting data. The transmission ends on the next start of frame or field boundary.

   Note: For Clocked Video Input IP, the frame or field matches the **Field order** parameter settings.

The starting and stopping of the IP synchronize to a frame or field boundary.

Table 17. Synchronization Settings for Clocked Video Input IP

The table lists the output of the IP with the different **Field order** settings.

Video Format	Field Order	Output
Interlaced	F1 first	Start, F1, F0, ..., F1, F0, Stop
Interlaced	F0 first	Start, F0, F1, ..., F0, F1, Stop
Interlaced	Any field first	Start, F0 or F1, ... F0 or F1, Stop
Progressive	F1 first	No output
Progressive	F0 first	Start, F0, F0, ..., F0, F0, Stop
Progressive	Any field first	Start, F0, F0, ..., F0, F0, Stop

7.3. Clocked Video Input IP Format Detection

The IP detects the format of the incoming clocked video and use it to create the Avalon streaming video control packet. The IP also provides this information in a set of registers.

Table 18. Format Detection

The IP can detect different aspects of the incoming video stream.

Format	Description
Picture width (in samples)	• The IP counts the total number of samples per line, and the number of samples in the active picture period. • One full line of video is required before the IP can determine the width.
Picture height (in lines)	• The IP counts the total number of lines per frame or field, and the number of lines in the active picture period. • One full frame or field of video is required before the IP can determine the height.
Interlaced/Progressive	• The IP detects whether the incoming video is interlaced or progressive. • If it is interlaced, separate height values are stored for both fields. • One full frame or field of video and a line from a second frame or field are required before the IP can determine whether the source is interlaced or progressive.
Standard	• The IP provides the contents of the `vid_std` bus through the Standard register. • When connected to the `rx_std` signal of an SDI IP instance, for example, these values can be used to report the standard (SD, HD, 3G, 6G, or 12G) of the incoming video. *Note:* In 3G, 6G, and 12G modes, the IP only supports YCbCr input color format with either 8 or 10 bits for each component.

7.3.1. Format Detection in Clocked Video Input II

After reset, if the CVI II IP has not determined the format of the incoming video, it uses the values you specify in the **Avalon-ST Video Initial/Default Control Packet** section in the parameter editor.

When the IP detects a resolution, it uses the resolution to generate the Avalon streaming viideo control packets until a new resolution is detected.

When the resolution valid bit in the `Status` register is 1, the `Active Sample Count`, `F0 Active Line Count`, `F1 Active Line Count`, `Total Sample Count`, `F0 Total Line Count`, `F1 Total Line Count`, and `Standard` registers are valid and contain readable values. The interlaced bit of the `Status` register is also valid and you can read it.

7.3.2. Interrupts

The Clocked Video Input IP produces a single interrupt line.

Table 19. Internal Interrupts

The table lists the internal interrupts of the interrupt line.

IP Core	Internal Interrupts	Description
Clocked Video Input II IP	Status update interrupt	Triggers when the stable bit, the resolution valid bit, the overflow sticky bit, or the picture drop sticky bit of the `Status` register changes value.
	End of field/frame interrupt	• If the synchronization settings are set to **Any field first**, triggers on the falling edge of the v sync. • If the synchronization settings are set to **F1 first**, triggers on the falling edge of the F1 v sync. • If the synchronization settings are set to **F0 first**, triggers on the falling edge of the F0 v sync. You can use this interrupt to trigger the reading of the ancillary packets from the control interface before the packets are overwritten by the next frame.

You can independently enable these interrupts using bits [2:1] of the `Control` register. You can read the interrupt values using bits [2:1] of the `Interrupt` register. Writing 1 to either of these bits clears the respective interrupt.

7.4. Clocked Video Output IP Video Modes

The video frame is described using the mode registers that you access through the Avalon memory-mapped control port.

If you turn off **Use control port** in the parameter editor for the IP, the output video format always has the format specified in the parameter editor.

You can configure the IP to support between 1 to 13 different modes and each mode has a bank of registers that describe the output frame.

When the IP receives a new control packet on the Avalon streaming video input, it searches the mode registers for a mode that is valid. The valid mode must have a field width and height that matches the width and height in the control packet.

The `Video Mode Match` register shows the selected mode:

- If the IP finds a matching mode, it completes the current frame; duplicating data if needed before commencing output with the new settings at the beginning of the next frame.
- If the IP does not find a matching mode, the video output format is unchanged.

If a new control packet is encountered before the expected end of frame, the IP completes the timing of the current frame with the remaining pixels taking the value of the last pixel output. The IP changes modes to the new packet at the end of this frame, unless you enable the Low Latency mode. During this period, when the FIFO buffer fills, the IP back-pressures the input until it is ready to transmit the new frame.

You must enable the `Go` bit to program the mode control registers. The sync signals, controlled by the mode control registers, reside in the video clock domain. The register control interface resides in the streaming clock domain. Enabling the `Go` bit, indicating that both clocks are running, avoids situations where a write in the streaming side cannot be issued to the video clock side because the video clock isn't running.

Figure 38. Progressive Frame Parameters

The figure shows how the register values map to the progressive frame format.

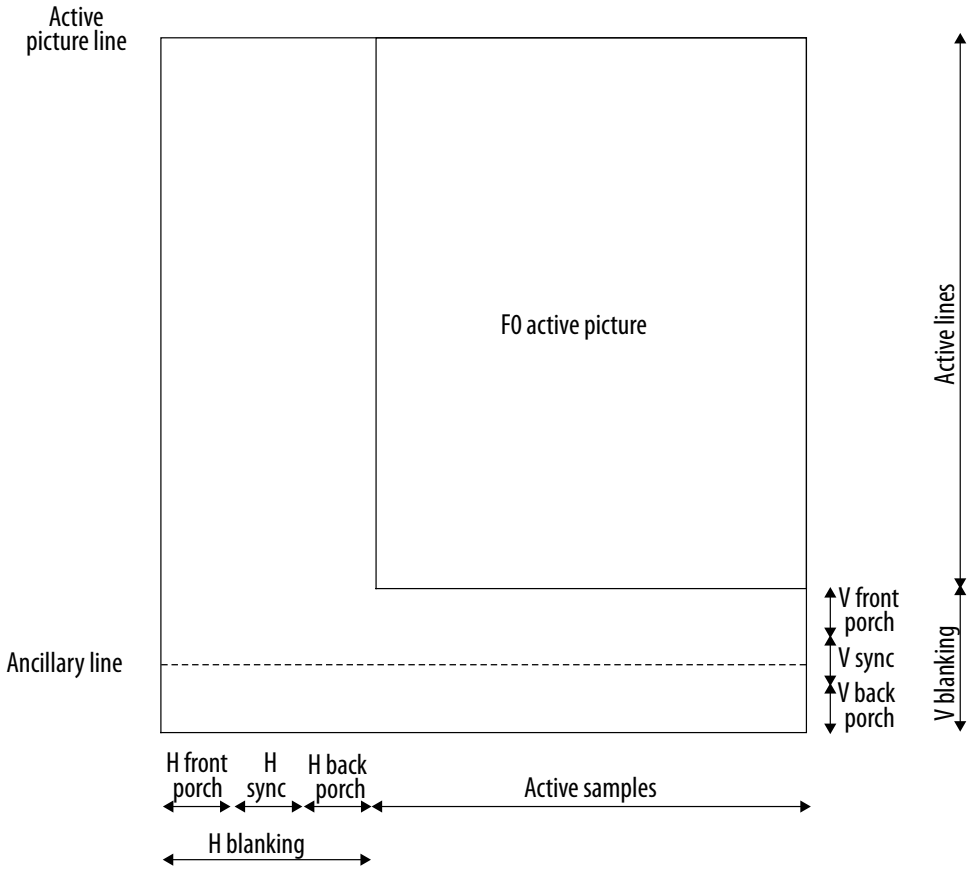
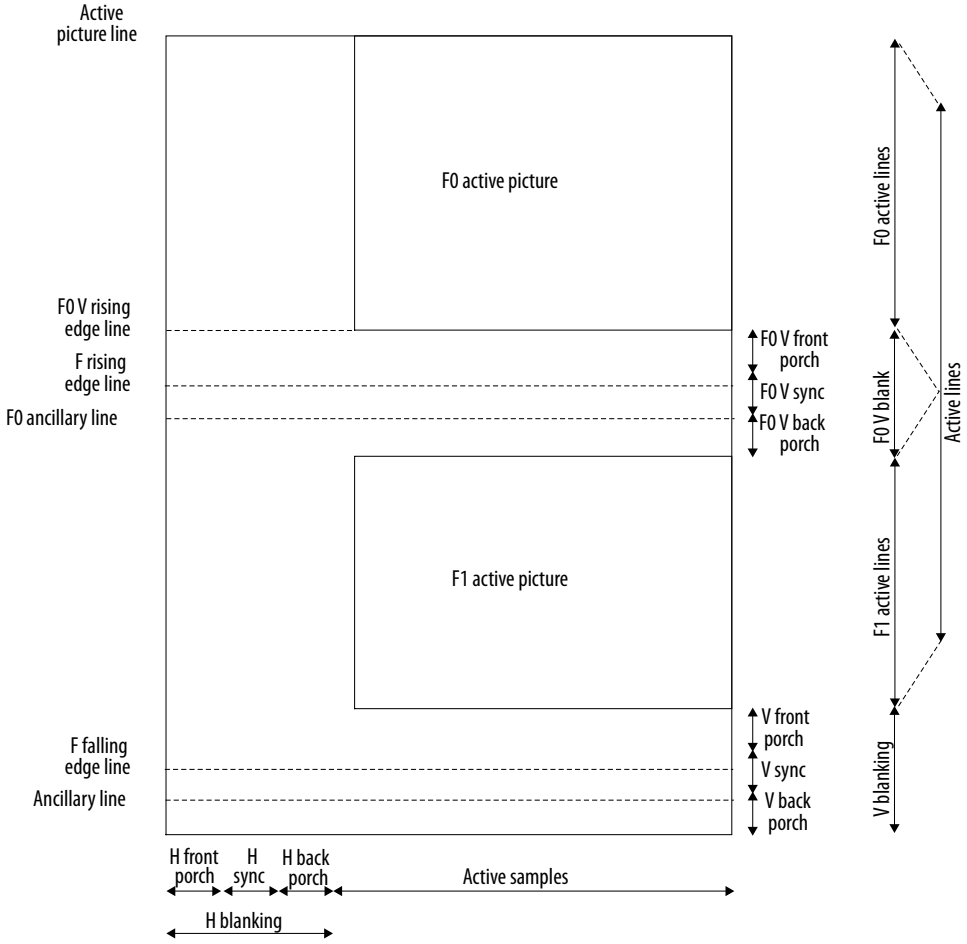


Figure 39. Interlaced Frame Parameters

The figure shows how the register values map to the interlaced frame format.



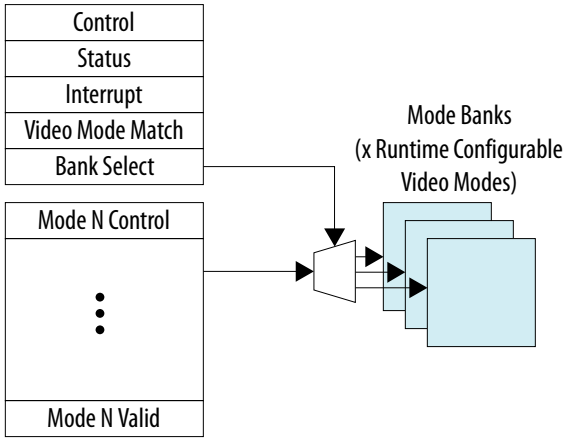
You can only write to the mode register of a mode bank if that mode is marked as invalid. To reconfigure mode 1:

1. Write 1 to the `Bank Select` register.
2. Write 0 to the `Mode N Valid` configuration register.
3. Write to the Mode N configuration registers, the Clocked Video Output II IP mirrors these writes internally to the selected bank.
4. Write 1 to the `Mode N Valid` register. The mode is now valid and you can select it.

You can configure a currently-selected mode in this way without affecting the video output of the IP.

If multiple modes match the resolution, the function selects the lowest mode. For example, the function selects `Mode1` over `Mode2` if both modes match. To allow the function to select `Mode2`, invalidate `Mode1` by writing a 0 to its mode valid register. Invalidating a mode does not clear its configuration.

Figure 40. Mode Bank Selection



7.4.1. Interrupts

The Clocked Video Output IP produces a single interrupt line.

This interrupt line is the OR of the following internal interrupts:

- Status update interrupt—triggers when the IP updates the `Video Mode Match` register when you select a new video mode.
- Locked interrupt—triggers when the outgoing video SOF aligns to the incoming SOF.

You can independently enable both interrupts using bits [2:1] of the `Control` register. You can read their values using bits [2:1] of the `Interrupt` register. Writing 1 to either of these bits clears the respective interrupt.

7.5. Clocked Video Output II Latency Mode

The Clocked Video Output II IP provides a low latency mode..

You can enable the low latency mode by setting the **Low latency mode** parameter to 1 in the Clocked Video Output II parameter editor. In the low latency mode, when the IP sees an early end of frame, or a change of resolution, the IP immediately updates the selected mode, resets the internal counters, and starts transmitting the new frame. These changes happen even if the timing is only part way through the previous frame.

If you choose not to enable the low latency mode, an early end of packet or change of resolution pauses reading of the FIFO buffer until the timing of the current frame is completed. Only at this point, the IP updates any new timings and starts transmitting new frames. In this mode, if the IP receives a partial video frame , when the FIFO buffer has filled, it generates backpressure until the current frame (including vertical sync) has completed.

7.6. Generator Lock

Generator lock (Genlock) locks the timing of video outputs to a reference source. Sources that are locked to the same reference can be cleanly switched between on a frame boundary.

For Clocked Video Input II IP, the `refclk_div` signal is a pulse on the rising edge of the H sync which a PLL can align its output clock to.

Clocked Video Input II IP

For the Clocked Video Input II IP, the SOF signal produces a pulse on the rising edge of the V sync. For interlaced video, the pulse is only produced on the rising edge of the F0 field, not the F1 field. The IP indicates a start of frame by a rising edge on the SOF signal (0 to 1).

7.7. Underflow and Overflow

Moving between the domain of clocked video and the flow controlled Avalon streaming video can cause flow problems. The Clocked Video Interface IP contains a FIFO buffer that accommodates any bursts in the flow data when set to a large enough value. The FIFO buffer can accommodate any bursts as long as the input or output rate of the upstream or downstream Avalon streaming video components is equal to or higher than that of the incoming/outgoing clocked video.

Underflow

The FIFO buffer can accommodate any bursts if the output rate of the downstream Avalon streaming video components is equal to or higher than that of the outgoing clocked video. If not , the FIFO underflows. If underflow occurs, the CVO IP continues to produce video and resynchronizing the `startofpacket` for the next image packet, from the Avalon streaming video interface with the start of the next frame. You can detect the underflow by looking at bit 2 of the `Status` register. This bit is sticky and if an underflow occurs, it stays at 1 until the bit is cleared by writing a 1 to it.

Overflow

The FIFO buffer can accommodate any bursts if the input rate of the upstream Avalon streaming video components is equal to or higher than that of the incoming clocked video. If not, the FIFO overflows. If overflow occurs, the IP produces an early `endofpacket` signal to complete the current frame. It then waits for the next start of frame (or field) before resynchronizing to the incoming clocked video and beginning to produce data again. The overflow is recorded in bit [9] of the `Status` register. This bit is sticky, and if an overflow occurs, it stays at 1 until the bit is cleared by writing a 0 to it. In addition to the overflow bit, you can read the current level of the FIFO buffer from the `Used Words` register.

The IP uses the height and width parameters at the point the frame was completed early in the control packet of the subsequent frame. If you are reading back the detected resolution, these unusual resolution values can make the IP seem to be operating incorrectly. When actually, the downstream system is failing to service the CVI IP at the necessary rate.

7.8. Timing Constraints

You need to constrain the Clocked Video Input II and Clocked Video Output II IPs.

For these IPs, the `.sdc` files are automatically included by their respective `.qip` files. After adding the Platform Designer system to your design in the Intel Quartus Prime software, verify that the `alt_vip_cvi_core.sdc` or `alt_vip_cvo_core.sdc` has been included.

When you apply the `.sdc` file, you may see some warning messages similar to the format below:

- Warning: At least one of the filters had some problems and could not be matched.
- Warning: * could not be matched with a keeper.

You should expect these warnings, because in certain configurations the Intel Quartus Prime software optimizes unused registers and they no longer remain in your design.

Intel recommends that you place a frame buffer in any CVI to CVO system. Because the CVO II IP generates sync signals for a complete frame, even when video frames end early, the CVO II IP can continually generate backpressure to the CVI II IP so that it keeps ending packets early.

Placing a frame buffer may not be appropriate if the system requires latency lower than 1 frame. In this case, enable the Low Latency mode when you configure the CVO II IP.

7.9. Handling Ancillary Packets

The Clocked Video Interface IPs use active format description (AFD) extractor and inserter examples to handle ancillary packets.

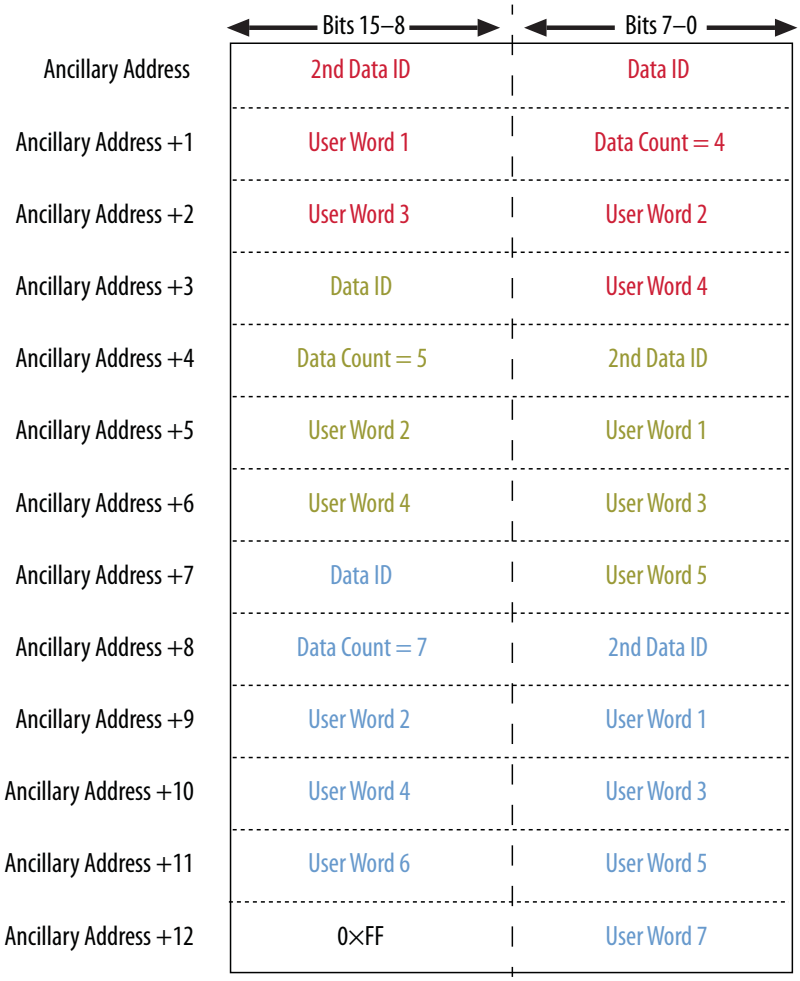
Ancillary Packets (Clocked Video Input II IP)

When you turn on the **Extract Ancillary Packets** parameter in embedded sync mode, the Clocked Video Input IP extracts any ancillary packets that are present in the Y channel of the incoming video's vertical blanking. The ancillary packets are stripped of their TRS code and placed in a RAM. You can access these packets by reading from the `Ancillary Packet` register. The packets are packed end to end from their Data ID to their final user word.

The RAM is 16 bits wide—two 8-bit ancillary data words are packed at each address location. The first word is at bits 0–7 and the second word is at bits 8–15. A word of all 1's indicates that no further ancillary packets are present and can appear in either the first word position or the second word position.

Figure 41. Ancillary Packet Register

The figure shows the position of the ancillary packets. The different colors indicate different ancillary packets.



Use the **Depth of ancillary memory** parameter to control the depth of the ancillary RAM. If available space is insufficient for all the ancillary packets, excess packets will be lost. The IP fills the ancillary RAM from the lowest memory address to the highest during each vertical blanking period—the IP overwrites the packets from the previous blanking periods. To avoid missing ancillary packets, the ancillary RAM should be read every time the `End of field/frame interrupt` register triggers.

AFD Inserter (Clocked Video Output IP)

When the output of the AFD Inserter connects to the input of the Clocked Video Output IP, the AFD Inserter inserts an Avalon streaming video ancillary data packet into the stream after each control packet. The AFD Inserter sets the DID and SDID of the ancillary packet to make it an AFD packet (DID = 0x41, SDID = 0x5). The contents of the ancillary packet are controlled by the AFD Inserter register map.

You can get the AFD Extractor from `<install_dir>\ip\altera\clocked_video_output\afd_example`.

Table 20. AFD Inserter Register Map

Address	Register	Description
0	Control	• When bit 0 is 0, the core discards all packets. • When bit 0 is 1, the core passes through all non-ancillary packets.
1	—	Reserved.
2	—	Reserved.
3	AFD	Bits 0-3 contain the active format description code.
4	AR	Bit 0 contains the aspect ratio code.
5	Bar data flags	Bits 0-3 contain the bar data flags to insert.
6	Bar data value 1	Bits 0-15 contain bar data value 1 to insert.
7	Bar data value 2	Bits 0-15 contain bar data value 2 to insert.
8	AFD valid	• When bit 0 is 0, an AFD packet is not present for each image packet. • When bit 0 is 1, an AFD packet is present for each image packet.

7.10. Modules for Clocked Video Input II IP Core

The architecture for the Clocked Video Input II IP differs from the previous clocked video input IP.

Figure 42. Block Diagram for Clocked Video Input II IP

The figure shows a block diagram of the Clocked Video Input II IP architecture.

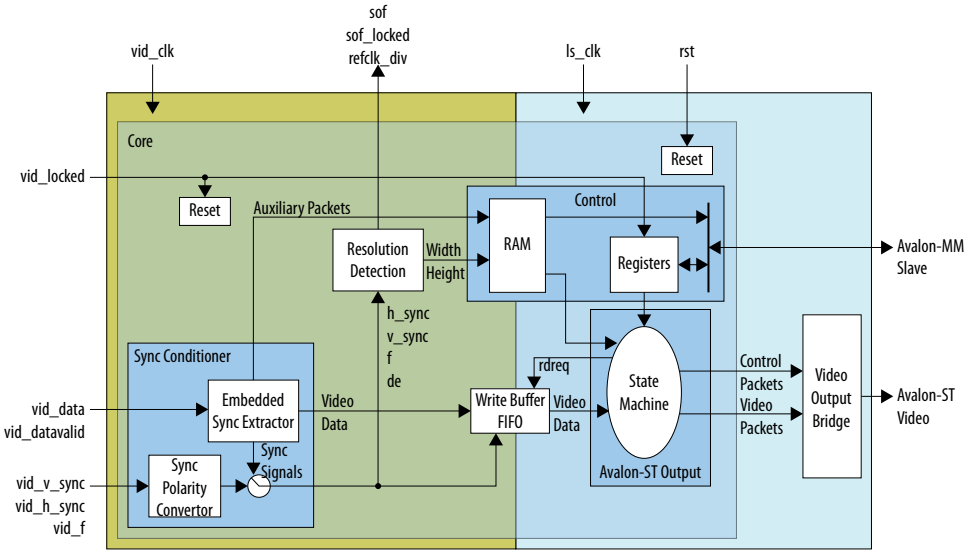


Table 21. Modules for Clocked Video Input II IP

The table below describes the modules in the Clocked Video Input II IP architecture.

Modules	Description
Sync_conditioner	• In embedded sync mode, this module extracts the embedded syncs from the video data and produces h_sync, v_sync, de, and f signals. • The module also extracts any ancillary packets from the video and writes them into a RAM in the control module. • In separate sync modes, this module converts the incoming sync signals to active high and produces h_sync, v_sync, de, and f signals. • If you turn on the **Extract field signal** parameter, the f signal is generated based on the position of the V-sync. If the rising edge of the V-sync occurs when h_sync is high, the f signal is set to 1, otherwise it is set to 0.
Resolution_detection	• This module uses the h_sync, v_sync, de, and f signals to detect the resolution of the incoming video. • The resolution consists of: — width of the line — width of the active picture region of the line (in samples) — height of the frame (or fields in the case of interlaced video) — height of the active picture region of the frame or fields (in lines) The IP writes the resolutions into a RAM in the control module. • The resolution detection module also produces some additional information. • It detects whether the video is interlaced by looking at the f signal. It detects whether the video is stable by comparing the length of the lines. If two outputs of the last three lines have the same length. the video is considered stable. • Finally, it determines if the resolution of the video is valid by checking that the width and height of the various regions of the frame has not changed.
Write_buffer_fifo	• This module writes the active picture data, marked by the de signal, into a FIFO that is used to cross over into the is_clk clock domain. • If you set the **Color plane transmission format** parameter to **Parallel** for the output, the write_buffer_fifo also converts any incoming sequential video, marked by the hd_sdn signal, into parallel video before writing it into the FIFO buffer. • The Go bit of the Control register must be 1 on the falling edge of the v_sync signal before the write_buffer_fifo module starts writing data into the FIFO buffer. • If an overflow occurs because of insufficient room in the FIFO buffer, the module stops writing active picture data into the FIFO buffer. • It waits for the start of the next frame before attempting to write in video data again.
Control	• This module provides the register file that is used to control the IP through an Avalon memory-mapped target interface. • It also holds the RAM that contains the detected resolution of the incoming video and the extracted auxiliary packet which is read by the av_st_output module, to form the control packets, and can also be read from the Avalon memory-mapped target interface. • The RAM provides the clock crossing between the vid_clk and is_clk clock domains.
Av_st_output	• This module creates the control packets, from the detected resolution read from the control module, and the video packets, from the active picture data read from the write_buffer_fifo module. • The IP sends the packets to the Video Output Bridge which turns them into Avalon streaming video packets.
Sdi_resampler	• This module is present only when you turn on the **Support 6G and 12G-SDI** parameter in the parameter editor. • It reformats the video stream from 2SI 6G- and 12G-SDI formats into the Avalon streaming video raster format and reformats the video stream from 3G-, HD-, and SD-SDI formats into the Avalon streaming video format. *Note:* This module reformats the SDI video streams only for configurations with 4 pixels in parallel.

7.11. Clocked Video Input II Signals, Parameters, and Registers

7.11.1. Clocked Video Input II Interface Signals

Table 22. Clocked Video Input II Signals

Signal	Direction	Description
main_reset_reset	Input	The IP core asynchronously resets when you assert this signal. You must deassert this signal synchronously to the rising edge of the clock signal.
main_clock_clk	Input	The main system clock. The IP core operates on the rising edge of this signal.
dout_data	Output	dout port Avalon-ST data bus. This bus enables the transfer of pixel data out of the IP core.
dout_endofpacket	Output	dout port Avalon-ST endofpacket signal. This signal is asserted when the IP core is ending a frame.
dout_ready	Input	dout port Avalon-ST ready signal. The downstream device asserts this signal when it is able to receive data.
dout_startofpacket	Output	dout port Avalon-ST startofpacket signal. This signal is asserted when the IP core is starting a new frame.
dout_valid	Output	dout port Avalon-ST valid signal. This signal is asserted when the IP core produces data.
dout_empty	Output	dout port Avalon-ST empty signal. This signal has a non-zero value only when you set the **Number of pixels in parallel** parameter to be greater than 1. This signal specifies the number of pixel positions which are empty at the end of the dout_endofpacket signal.
status_update_int	Output	control slave port Avalon-MM interrupt signal. When asserted, the status registers of the IP core have been updated and the master must read them to determine what has occurred. *Note:* Present only if you turn on **Use control port**.
vid_clk	Input	Clocked video clock. All the video input signals are synchronous to this clock.
vid_data	Input	Clocked video data bus. This bus enables the transfer of video data into the IP core.
vid_de	Input	Clocked video data enable signal. The driving core asserts this signal to indicate that the data on vid_data is part of the active picture region of an incoming video. This signal must be driven for correct operation of the IP core. *Note:* For separate synchronization mode only.
vid_datavalid	Input	Enabling signal for the CVI II IP core. The IP core only reads the vid_data, vid_de, vid_h_sync, vid_v_sync, vid_std, and vid_f signals when vid_datavalid is 1. This signal allows the CVI II IP core to support oversampling during when the video runs at a higher rate than the pixel clock. *Note:* If you are not oversampling your input video, tie this signal high.
vid_locked	Input	Clocked video locked signal. Assert this signal when a stable video stream is present on the input. Deassert this signal when the video stream is removed. When 0, this signal triggers an early end of output frame packet and does not reset the internal registers. When this signal recovers after 0, if the system is not reset from outside, the first frame may have leftover pixels from the lock-lost frame,

continued...

Signal	Direction	Description
vid_f	Input	Clocked video field signal. For interlaced input, this signal distinguishes between field 0 and field 1. For progressive video, you must deassert this signal. *Note:* For separate synchronization mode only.
vid_v_sync	Input	Clocked video vertical synchronization signal. Assert this signal during the vertical synchronization period of the video stream. *Note:* For separate synchronization mode only.
vid_h_sync	Input	Clocked video horizontal synchronization signal. Assert this signal during the horizontal synchronization period of the video stream. *Note:* For separate synchronization mode only.
vid_hd_sdn	Input	Clocked video color plane format selection signal . This signal distinguishes between sequential (when low) and parallel (when high) color plane formats. *Note:* For run-time switching of color plane transmission formats mode only.
vid_std	Input	Video standard bus. Can be connected to the rx_std signal of the SDI IP core (or any other interface) to read from the Standard register.
vid_color_encoding	Input	This signal is captured in the Color Pattern register and does not affect the functioning of the IP core. It provides a mechanism for control processors to read incoming color space information if the IP core (e.g. HDMI RX core) driving the CVI II does not provide such an interface. Tie this signal to low if no equivalent signal is available from the IP core driving CVI II.
vid_bit_width	Input	This signal is captured in the Color Pattern register and does not affect the functioning of the IP core. It provides a mechanism for control processors to read incoming video bit width information if the IP core (e.g. HDMI RX core) driving the CVI II does not provide such an interface. Tie this signal to low if no equivalent signal is available from the IP core driving CVI II.
vid_total_sample_count	Input	The IP core creates this signal if you do not turn on the **Extract the total resolution** parameter. The CVI II IP core operates using this signal as the total horizontal resolution instead of an internally detected version.
Vid_total_line_count	Input	The IP core creates this signal if you do not turn on the **Extract the total resolution** parameter. The CVI II IP core operates using this signal as the total vertical resolution instead of an internally detected version.
sof	Output	Start of frame signal. A change of 0 to 1 indicates the start of the video frame as configured by the SOF registers. Connecting this signal to a CVO IP core allows the function to synchronize its output video to this signal.
sof_locked	Output	Start of frame locked signal. When asserted, the sof signal is valid and can be used.
refclk_div	Output	A single cycle pulse in-line with the rising edge of the h sync.
clipping	Output	Clocked video clipping signal. A signal corresponding to the clipping bit of the Status register synchronized to vid_clk. This signal is for information only and no action is required if it is asserted.
padding	Output	Clocked video padding signal. A signal corresponding to the padding bit of the Status register synchronized to vid_clk. This signal is for information only and no action is required if it is asserted.
overflow	Output	Clocked video overflow signal. A signal corresponding to the overflow sticky bit of the Status register synchronized to vid_clk. This signal is for information only and no action is required if it is asserted.

continued...

Signal	Direction	Description
		Note: Present only if you turn on **Use control port**.
vid_hdmi_duplication[3:0]	Input	If you select **Remove duplicate pixels** in the parameter, this 4-bit bus is added to the CVI II interface. You can drive this bus based on the number of times each pixel is duplicated in the stream (HDMI-standard compliant).

Table 23. Control Signals for CVI II IP Cores

Signal	Direction	Description
av_address	Input	control slave port Avalon-MM address bus. Specifies a word offset into the slave address space. *Note:* Present only if you turn on **Use control port**.
av_read	Input	control slave port Avalon-MM read signal. When you assert this signal, the control port drives new data onto the read data bus. *Note:* Present only if you turn on **Use control port**.
av_readdata	Output	control slave port Avalon-MM read data bus. These output lines are used for read transfers. *Note:* Present only if you turn on **Use control port**.
av_waitrequest	Output	control slave port Avalon-MM wait request bus. This signal indicates that the slave is stalling the master transaction. *Note:* Present only if you turn on **Use control port**.
av_write	Input	control slave port Avalon-MM write signal. When you assert this signal, the control port accepts new data from the write data bus. *Note:* Present only if you turn on **Use control port**.
av_writedata	Input	control slave port Avalon-MM write data bus. These input lines are used for write transfers. *Note:* Present only if you turn on **Use control port**.
av_byteenable	Input	control slave port Avalon-MM byteenable bus. These lines indicate which bytes are selected for write and read transactions.

7.11.2. Clocked Video Input II Parameter Settings

Table 24. Clocked Video Input II Parameter Settings

Parameter	Value	Description
Bits per pixel per color plane	4–20, Default = **8**	Select the number of bits per pixel (per color plane).
Number of color planes	1–4, Default = **3**	Select the number of color planes.
Color plane transmission format	• Sequence • **Parallel**	Specify whether to transmit the color planes in sequence or in parallel. If you select multiple pixels in parallel, then select **Parallel**.
Number of pixels in parallel	**1**, 2, 4, or 8	Specify the number of pixels transmitted or received in parallel.
Field order	• **Field 0 first** • Field 1 first • Any field first	Specify the field to synchronize first when starting or stopping the output.
Enable matching data packet to control by clipping	On or **Off**	When there is a change in resolution,the control packet and video data packet transmitted by the IP core mismatch. Turn on this parameter if you want to clip the input video frame to match the resolution sent in control packet.

continued...

Parameter	Value	Description
		When the current input frame resolution is wider and/or taller than the one specified in the control packet, then the IP core clips them to match the control packet dimensions.
Enable matching data packet to control by padding	On or **Off**	Turn on this parameter if you also want to pad the incoming frame if it is smaller and/or shorter than the resolution specified in the control packet. *Note:* This parameter is available only when you turn on **Enable matching data packet to control by clipping**. Depending on the size of the mismatch, padding operation could lead to frame drops at the input.
Overflow handling	On or **Off**	Turn this parameter if you want the to finish the current frame (with dummy pixel data) based on the resolution specified in the control packet if overflow happens. The IP core waits for the FIFO to become empty before it starts the padding process. By default (turned off), if an overflow is encountered, current frame is terminated abruptly. *Note:* Depending on size of the frame left to finish and the back pressure from downstream IP, overflow handling operation could lead to frame drops at the input.
Sync signals	• Embedded in video • **On separate wires**	Specify whether to embed the synchronization signal in the video stream or provide on a separate wire.
Support 6G and 12G-SDI	On or **Off**	Turn on to enable 6G-SDI or 12G-SDI support for CVI II IP core. Turning on this option will fix the number of pixels in parallel to 4. *Note:* This option is available only when you select the **Embedded in video** option for the **Sync signals** parameter.
Allow color planes in sequence input	On or **Off**	Turn on if you want to allow run-time switching between sequential and parallel color plane transmission formats. The format is controlled by the `vid_hd_sdn` signal.
Extract field signal	On or **Off**	Turn on to internally generate the field signal from the position of the V sync rising edge.
Use vid_std bus	On or **Off**	Turn on if you want to use the video standard, `vid_std`. *Note:* Platform Designer always generates the `vid_std` signal even when you turn off this parameter. The IP core samples and stores this signal in the `Standard` register to be read back for software control. If not needed, leave this signal disconnected.
Width of vid_std bus	External sync: 1–16, Default = **1** Embedded sync: 3, Default = **3**	Specify the width of the `vid_std` bus, in bits.
Extract ancillary packets	On or **Off**	Turn on to extract the ancillary packets in embedded sync mode.
Depth of the ancillary memory	0–4096, Default = **0**	Specify the depth of the ancillary packet RAM, in words.
Extract the total resolution	**On** or Off	Turn on to extract total resolution from the video stream.

continued...

Parameter	Value	Description
Enable HDMI duplicate pixel removal	• No duplicate pixel removal • Remove duplicate pixel	Specify whether to enable a block to remove duplicate pixels for low rate resolutions. When you select **Remove duplicate pixel**, the IP core generates an additional 4-bit port to connect to the HDMI IP core. This port extracts the duplication factor from the HDMI IP core. *Note:* The CVI II IP core currently supports only duplication factors of 0 (no duplication) or 1 (each pixel transmitted twice).
Interlaced or progressive	• **Progressive** • Interlaced	Specify the format to be used when no format is automatically detected.
Width	32–8192, Default = **1920**	Specify the image width to be used when no format is automatically detected.
Height – frame/field 0	32–8192, Default = **1080**	Specify the image height to be used when no format is automatically detected.
Height – field 1	32–8192, Default = **480**	Specify the image height for interlaced field 1 to be used when no format is automatically detected.
Pixel FIFO size	32–4096, Default = **2048**	Specify the required FIFO depth in pixels.
Video in and out use the same clock	On or **Off**	Turn on if you want to use the same signal for the input and output video image stream clocks.
Use control port	On or **Off**	Turn on to use the optional stop/go control port.

7.11.3. Clocked Video Input II Control Registers

Table 25. Clocked Video Input II Registers

Address	Register	Description
0	`Control`	• Bit 0 of this register is the Go bit. Setting this bit to 1 causes the CVI II IP core to start data output on the next video frame boundary. • Bits 3, 2, and 1 of the `Control` register are the interrupt enables: — Setting bit 1 to 1, enables the status update interrupt. — Setting bit 2 to 1, enables the end of field/frame video interrupt.
1	`Status`	• Bit 0 of this register is the `Status` bit. This bit is asserted when the CVI IP core is producing data. • Bits 6–1 of the `Status` register are unused. • Bit 7 is the interlaced bit. When asserted, the input video stream is interlaced. • Bit 8 is the stable bit. When asserted, the input video stream has had a consistent line length for two of the last three lines. • Bit 9 is the overflow sticky bit. When asserted, the input FIFO has overflowed. The overflow sticky bit stays asserted until a 1 is written to this bit. • Bit 10 is the resolution bit. When asserted, indicates a valid resolution in the sample and line count registers. • Bit 11 is the `vid_locked` bit. When asserted, indicates current signal value of the `vid_locked` signal. • Bit 12 is the clipping bit. When asserted, input video frame/field is being clipped to match the resolution specified in the control packet. *Note:* Present only when you turn on **Enable matching data packet to control by clipping**.

continued...

Address	Register	Description
		• Bit 13 is the padding bit. When asserted, input video frame/field is being padded to match the resolution specified in the control packet. *Note:* Present only when you turn on **Enable matching data packet to control by padding**. • Bit 14 is the picture drop sticky bit. When asserted, indicates one or more picture(s) has been dropped at input side. It stays asserted until a 1 is written to this bit. • Bits 21–15 give the picture drop count. When picture drop sticky bit is asserted, this drop count provides the number of frame/field dropped at the input. Count resets when you clear the picture drop sticky bit. *Note:* Both picture drop sticky and picture drop count bit are present only when you turn on **Enable matching data packet to control by padding** and/or **Overflow handling**.
2	`Interrupt`	Bits 2 and 1 are the interrupt status bits: • When bit 1 is asserted, the status update interrupt has triggered. • When bit 2 is asserted, the end of field/frame interrupt has triggered. • The interrupts stay asserted until a 1 is written to these bits.
3	`Used Words`	The used words level of the input FIFO.
4	`Active Sample Count`	The detected sample count of the video streams excluding blanking.
5	`F0 Active Line Count`	The detected line count of the video streams F0 field excluding blanking.
6	`F1 Active Line Count`	The detected line count of the video streams F1 field excluding blanking.
7	`Total Sample Count`	The detected sample count of the video streams including blanking.
8	`F0 Total Line Count`	The detected line count of the video streams F0 field including blanking.
9	`F1 Total Line Count`	The detected line count of the video streams F1 field including blanking.
10	`Standard`	The contents of the `vid_std` signal.
11-13	Reserved	Reserved for future use.
14	`Color Pattern`	• Bits 7–0 are for color encoding—captures the value driven on the `vid_color_encoding` input. • Bits 15–8 are for bit width—captures the value driven on the `vid_bit_width` input.
15	`Ancillary Packet`	Start of the ancillary packets that have been extracted from the incoming video.
15 + Depth of ancillary memory		End of the ancillary packets that have been extracted from the incoming video.

7.12. Clocked Video Output II Signals, Parameters, and Registers

7.12.1. Clocked Video Output II Interface Signals

Table 26. Clocked Video Output II Signals

Signal	Direction	Description
`main_reset_reset`	Input	The IP core asynchronously resets when you assert this signal. You must deassert this signal synchronously to the rising edge of the clock signal.
`main_clock_clk`	Input	The main system clock. The IP core operates on the rising edge of this signal.

continued...

Signal	Direction	Description
din_data	Input	din port Avalon-ST data bus. This bus enables the transfer of pixel data into the IP core.
din_endofpacket	Input	din port Avalon-ST endofpacket signal. This signal is asserted when the downstream device is ending a frame.
din_ready	Output	din port Avalon-ST ready signal. This signal is asserted when the IP core function is able to receive data.
din_startofpacket	Input	din port Avalon-ST startofpacket signal. Assert this signal when the downstream device is starting a new frame.
din_valid	Input	din port Avalon-ST valid signal. Assert this signal when the downstream device produces data.
din_empty	Input	din port Avalon-ST empty signal. This signal has a non zero value only when you set the Number of pixels in parallel parameter to be greater than 1. This signal specifies the number of pixel positions which are empty at the end of the din_endofpacket signal.
underflow	Output	Clocked video underflow signal. A signal corresponding to the underflow sticky bit of the Status register synchronized to vid_clk. This signal is for information only and no action is required if it is asserted. *Note:* Present only if you turn on **Use control port**.
status_update_int	Output	control slave port Avalon-MM interrupt signal. When asserted, the status registers of the IP core have been updated and the master must read them to determine what has occurred. *Note:* Present only if you turn on **Use control port**.
vid_clk	Input	Clocked video clock. All the video output signals are synchronous to this clock.
vid_data	Output	Clocked video data bus. This bus transfers video data out of the IP core.
vid_datavalid	Output	Clocked video data valid signal. Assert this signal when a valid sample of video data is present on vid_data. *Note:* This signal is equivalent to the CVI II IP core's vid_de signal.
vid_f	Output	Clocked video field signal. For interlaced input, this signal distinguishes between field 0 and field 1. For progressive video, this signal is unused. *Note:* For separate synchronization mode only.
vid_h	Output	Clocked video horizontal blanking signal. This signal is asserted during the horizontal blanking period of the video stream. *Note:* For separate synchronization mode only.
vid_h_sync	Output	Clocked video horizontal synchronization signal. This signal is asserted during the horizontal synchronization period of the video stream. *Note:* For separate synchronization mode only.
vid_ln	Output	Clocked video line number signal. Used with the SDI IP core to indicate the current line number when the vid_trs signal is asserted. *Note:* For embedded synchronization mode only.
vid_mode_change	Output	Clocked video mode change signal. This signal is asserted on the cycle before a mode change occurs.
vid_std	Output	Video standard bus. Can be connected to the tx_std signal of the SDI IP core (or any other interface) to read from the Standard register.
vid_trs	Output	Clocked video time reference signal (TRS) signal. Used with the SDI IP core to indicate a TRS, when asserted. *Note:* For embedded synchronization mode only.

continued...

Signal	Direction	Description
vid_v	Output	Clocked video vertical blanking signal. This signal is asserted during the vertical blanking period of the video stream. *Note:* For separate synchronization mode only.
vid_v_sync	Output	Clocked video vertical synchronization signal. This signal is asserted during the vertical synchronization period of the video stream. *Note:* For separate synchronization mode only.
vcoclk_div	Output	A divided down version of vid_clk (vcoclk). Setting the Vcoclk Divider register to be the number of samples in a line produces a horizontal reference on this signal. A PLL uses this horizontal reference to synchronize its output clock. *Note:* Present only if you turn on **Use control port**.
vid_sof	Output	Start of frame signal. A rising edge (0 to 1) indicates the start of the video frame as configured by the SOF registers. *Note:* Present only if you turn on **Use control port**.
vid_sof_locked	Output	Start of frame locked signal. When asserted, the vid_sof signal is valid and can be used. *Note:* Present only if you turn on **Use control port**.
sof	Input	Start of frame signal. A rising edge (0 to 1) indicates the start of the video frame as configured by the SOF registers. Connecting this signal to a CVI IP core allows the output video to be synchronized to this signal. *Note:* Present only if you turn on **Accept synchronization inputs**.
sof_locked	Input	Start of frame locked signal. When asserted, the sof signal is valid and can be used. *Note:* Present only if you turn on **Accept synchronization inputs**.
sdi_cvo_rden	Input	This signal connects to the SDI II IP core's tx_dataout_valid output signal. This signal indicates to the CVO II IP core to advance the state of the clocked video data and attributes. *Note:* This signal is only available when you select the **Embedded in video** option for the **Sync signals** parameter. It will not be present for external sync interfaces such as DisplayPort and HDMI.

Table 27. Control Signals for CVO II IP Cores

Signal	Direction	Description
av_address	Input	control slave port Avalon-MM address bus. Specifies a word offset into the slave address space. *Note:* Present only if you turn on **Use control port**.
av_read	Input	control slave port Avalon-MM read signal. When you assert this signal, the control port drives new data onto the read data bus. *Note:* Present only if you turn on **Use control port**.
av_readdata	Output	control slave port Avalon-MM read data bus. These output lines are used for read transfers. *Note:* Present only if you turn on **Use control port**.
av_waitrequest	Output	control slave port Avalon-MM wait request bus. This signal indicates that the slave is stalling the master transaction. *Note:* Present only if you turn on **Use control port**.
av_write	Input	control slave port Avalon-MM write signal. When you assert this signal, the control port accepts new data from the write data bus.

continued...

Signal	Direction	Description
		Note: Present only if you turn on **Use control port**.
av_writedata	Input	control slave port Avalon-MM write data bus. These input lines are used for write transfers. *Note:* Present only if you turn on **Use control port**.
av_byteenable	Input	control slave port Avalon-MM byteenable bus. These lines indicate which bytes are selected for write and read transactions.

7.12.2. Clocked Video Output II Parameter Settings

Table 28. Clocked Video Output II Parameter Settings

Parameter	Value	Description
Image width/Active pixels	32–8192, Default = **1920**	Specify the image width by choosing the number of active pixels.
Image height/Active lines	32–8192, Default = **1200**	Specify the image height by choosing the number of active lines.
Bits per pixel per color plane	4–20, Default = **8**	Select the number of bits per pixel (per color plane).
Number of color planes	1–4, Default = **3**	Select the number of color planes.
Color plane transmission format	• Sequence • **Parallel**	Specify whether to transmit the color planes in sequence or in parallel. If you select multiple pixels in parallel, then select **Parallel**.
Allow output of channels in sequence	On or **Off**	• Turn on if you want to allow run-time switching between sequential formats, such as NTSC, and parallel color plane transmission formats, such as 1080p. The format is controlled by the ModeXControl registers. • Turn off if you are using multiple pixels in parallel.
Number of pixels in parallel	**1**, 2, 4, or 8	Specify the number of pixels transmitted or received in parallel. *Note:* Number of pixels in parallel are only supported if you select **On separate wires** for the **Sync signals** parameter.
Interlaced video	On or **Off**	Turn off to use progressive video.
Sync signals	• Embedded in video • **On separate wires**	Specify whether to embed the synchronization signal in the video stream or to provide the synchronization signal on a separate wire. • Embedded in video: You can set the active picture line, horizontal blanking, and vertical blanking values. • On separate wires: You can set horizontal and vertical values for sync, front porch, and back porch.
Support 6G and 12G-SDI	On or **Off**	Turn on to enable 6G-SDI or 12G-SDI support for CVO II IP core. Turning on this option will fix the number of pixels in parallel to 4. *Note:* This option is available only when you select the **Embedded in video** option for the **Sync signals** parameter.

continued...

Parameter	Value	Description
Default SDI video standard	• SD • HD • **3GA** (3G-SDI Level A) • 6GB (6G-SDI with 8 streams interleaved) • 12GA (12G-SDI with 8 streams interleaved)	Specify the default SDI video standard. *Note:* This option is available only when you select the **Embedded in video** option for the **Sync signals** parameter. **6GB** and **12GA** options are available only when you turn on the **Support 6G and 12G-SDI** parameter.
Active picture line	32–65536, Default = **0**	Specify the start of active picture line for Frame.
Frame/Field 1: Ancillary packet insertion line	32–65536, Default = **0**	Specify the line where ancillary packet insertion starts.
Embedded syncs only - Frame/Field 1: Horizontal blanking	32–65536, Default = **0**	Specify the size of the horizontal blanking period in pixels for Frame/Field 1.
Embedded syncs only - Frame/Field 1: Vertical blanking	32–65536, Default = **0**	Specify the size of the vertical blanking period in pixels for Frame/Field 1.
Separate syncs only - Frame/Field 1: Horizontal sync	32–65536, Default = **44**	Specify the size of the horizontal synchronization period in pixels for Frame/Field 1.
Separate syncs only - Frame/Field 1: Horizontal front porch	32–65536, Default = **88**	Specify the size of the horizontal front porch period in pixels for Frame/Field 1.
Separate syncs only - Frame/Field 1: Horizontal back porch	32–65536, Default = **148**	Specify the size of the horizontal back porch in pixels for Frame/Field 1.
Separate syncs only - Frame/Field 1: Vertical sync	32–65536, Default = **5**	Specify the number of lines in the vertical synchronization period for Frame/Field 1.
Separate syncs only - Frame/Field 1: Vertical front porch	32–65536, Default = **4**	Specify the number of lines in the vertical front porch period in pixels for Frame/Field 1.
Separate syncs only - Frame/Field 1: Vertical back porch	32–65536, Default = **36**	Specify the number of lines in the vertical back porch in pixels for Frame/Field 1.
Interlaced and Field 0: F rising edge line	32–65536, Default = **0**	Specify the line when the rising edge of the field bit occurs for Interlaced and Field 0.
Interlaced and Field 0: F falling edge line	32–65536, Default = **0**	Specify the line when the falling edge of the field bit occurs for Interlaced and Field 0.
Interlaced and Field 0: Vertical blanking rising edge line	32–65536, Default = **0**	Specify the line when the rising edge of the vertical blanking bit for Field 0 occurs for Interlaced and Field 0.
Interlaced and Field 0: Ancillary packet insertion line	32–65536, Default = **0**	Specify the line where ancillary packet insertion starts.
Embedded syncs only - Field 0: Vertical blanking	32–65536, Default = **0**	Specify the size of the vertical blanking period in pixels for Interlaced and Field 0.
Separate syncs only - Field 0: Vertical sync	32–65536, Default = **0**	Specify the number of lines in the vertical synchronization period for Interlaced and Field 0.
Separate syncs only - Field 0: Vertical front porch	32–65536, Default = **0**	Specify the number of lines in the vertical front porch period for Interlaced and Field 0.
Separate syncs only - Field 0: Vertical back porch	32–65536, Default = **0**	Specify the number of lines in the vertical back porch period for Interlaced and Field 0.

continued...

Parameter	Value	Description
Pixel FIFO size	32–(memory limit), Default = **1920**	Specify the required FIFO depth in pixels, (limited by the available on-chip memory).
FIFO level at which to start output	0–(memory limit), Default = **1919**	Specify the fill level that the FIFO must have reached before the output video starts.
Video in and out use the same clock	On or **Off**	Turn on if you want to use the same signal for the input and output video image stream clocks.
Use control port	On or **Off**	Turn on to use the optional Avalon-MM control port.
Generate synchronization outputs	On or **Off**	When you turn on **Use control port**, this option becomes available. Turning on this option generates the `vid_vcoclk_div`, `vid_sof`, and `vid_sof_locked` output signals. You can use these signals to generate timing reference signals to synchronize video.
Accept synchronization inputs	On or **Off**	When you turn on **Generate synchronization outputs**, this option becomes available. Turning on this option generates the `sof` and `sof_locked` input signals. These signals enable the CVO II IP core to align the synchronization outputs to within 1 line of inputs.
Set vco_clk_divider increment to pixels in parallel	On or **Off**	When you turn on **Generate synchronization outputs**, this option becomes available. Turning on this option enables you to set `vco_clk_divider` to increment in 2 different modes: • Increment in single counts until the divider value is reached. • Increment in steps equal to the value specified in the `Number of pixels in parallel` parameter. The second increment allows `vco_clk_div` to keep step with the internal counters for the video output but introduces variation in the number of cycles between `divclk` pulses depending on the `divclk` value and how the value in the `Number of pixels in parallel` divides.
Low latency mode	0–1, Default = **0**	• Select 0 for regular completion mode. Each output frame initiated completes its timing before a new frame starts. • Select 1 for low latency mode. The IP core starts timing for a new frame immediately.
Run-time configurable video modes	1–13, Default = **1**	Specify the number of run-time configurable video output modes that are required when you are using the Avalon-MM control port. *Note:* This parameter is available only when you turn on **Use control port**.
Width of vid_std bus	External sync: 1–16, Default = **1** Embedded sync: 3, Default = **3**	Select the width of the `vid_std` bus, in bits.

7.12.3. Clocked Video Output II Control Registers

Note: If you configure the design without enabling the control interface, the interrupt line (`status_update_int`) will not be generated. This is because the logic required to clear the interrupt will not be generated and therefore could not provide useful information.

Table 29. Clocked Video Output II Registers

The rows in the table are repeated in ascending order for each video mode. All of the Mode*N* registers are write only.

Address	Register	Description
0	`Control`	• Bit 0 of this register is the Go bit. Setting this bit to 1 causes the CVO IP core to start video data output. • Bit 2 of the `Control` register is the `Clear Underflow Register` bit. When bit 2 of the `Status` register is set, a 1 should be written to this register to clear the underflow. • Bits 4 and 3 of the `Control` register are the `Genlock` control bits. — Setting bit 3 to 1 enables the synchronization outputs: `vid_sof`, `vid_sof_locked`, and `vcoclk_div`. — Setting bit 4 to 1, while bit 3 is 1, enables frame locking. The IP core attempts to align its `vid_sof` signal to the `sof` signal from the CVI IP core. • Bits 9 and 8 of the `Control` register are the interrupt enables, matching the position of the interrupt registers at address 2. — Setting bit 8 to 1 enables the status update interrupt. — Setting bit 9 to 1 enables the locked interrupt.
1	`Status`	• Bit 0 of this register is the `Status` bit. This bit is asserted when the CVO IP core is producing data. • Bit 1 of the `Status` register is unused. • Bit 2 is the underflow sticky bit. When bit 2 is asserted, the output FIFO has underflowed. The underflow sticky bit stays asserted until a 1 is written to bit 2 of the `Control` register • Bit 3 is the frame locked bit. When bit 3 is asserted, the CVO IP core has aligned its start of frame to the incoming `sof` signal.
2	`Interrupt`	Bits 9 and 8 are the interrupt status bits: • When bit 1 is asserted, the status update interrupt has triggered. • When bit 2 is asserted, the locked interrupt has triggered. • The interrupts stay asserted until a 1 is written to these bits.
3	`Video Mode Match`	Before any user specified modes are matched, this register reads back 0 indicating the default values are selected. Once a match has been made, the register reads back in a one-hot fashion, e.g. 0x0001=Mode0 0x00020=Mode5
4	`Bank Select`	Writes to the Mode*N* registers will be reflected to the mode bank selected by this register. Up to 13 banks are available depending on parameterization. Selection is by standard binary encoding.
5	`Mode`*N*` Control`	Video Mode*N* 1 Control. • Bit 0 of this register is the `Interlaced` bit. — Set to 1 for interlaced. — Set to 0 for progressive. • Bit 1 of this register is the sequential output control bit (only if the **Allow output of color planes in sequence** compile-time parameter is enabled). — Setting bit 1 to 1, enables sequential output from the CVO IP core (NTSC). — Setting bit 1 to 0, enables parallel output from the CVO IP core (1080p).
6	`Mode`*N*` Sample Count`	Video mode *N* sample count. Specifies the active picture width of the field.
7	`Mode`*N*` F0 Line Count`	Video mode *N* field 0/progressive line count. Specifies the active picture height of the field.

continued...

Address	Register	Description
8	`ModeN F1 Line Count`	Video mode *N* field 1 line count (interlaced video only). Specifies the active picture height of the field.
9	`ModeN Horizontal Front Porch`	Video mode *N* horizontal front porch. Specifies the length of the horizontal front porch in samples.
10	`ModeN Horizontal Sync Length`	Video mode *N* horizontal synchronization length. Specifies the length of the horizontal synchronization length in samples.
11	`ModeN Horizontal Blanking`	Video mode *N* horizontal blanking period. Specifies the length of the horizontal blanking period in samples.
12	`ModeN Vertical Front Porch`	Video mode *N* vertical front porch. Specifies the length of the vertical front porch in lines.
13	`ModeN Vertical Sync Length`	Video mode 1 vertical synchronization length. Specifies the length of the vertical synchronization length in lines.
14	`Mode1 Vertical Blanking`	Video mode *N* vertical blanking period. Specifies the length of the vertical blanking period in lines.
15	`ModeN F0 Vertical Front Porch`	Video mode *N* field 0 vertical front porch (interlaced video only). Specifies the length of the vertical front porch in lines.
16	`ModeN F0 Vertical Sync Length`	Video mode *N* field 0 vertical synchronization length (interlaced video only). Specifies the length of the vertical synchronization length in lines.
17	`ModeN F0 Vertical Blanking`	Video mode *N* field 0 vertical blanking period (interlaced video only). Specifies the length of the vertical blanking period in lines.
18	`ModeN Active Picture Line`	Video mode *N* active picture line. Specifies the line number given to the first line of active picture.
19	`ModeN F0 Vertical Rising`	Video mode *N* field 0 vertical blanking rising edge. Specifies the line number given to the start of field 0's vertical blanking.
20	`ModeN Field Rising`	Video mode *N* field rising edge. Specifies the line number given to the end of Field 0 and the start of Field 1.
21	`ModeN Field Falling`	Video mode *N* field falling edge. Specifies the line number given to the end of Field 0 and the start of Field 1.
22	`ModeN Standard`	The value output on the `vid_std` signal.
23	`ModeN SOF Sample`	Start of frame sample register. The sample and subsample upon which the SOF occurs (and the `vid_sof` signal triggers): • Bits 1–0 are the subsample value. • Bits 15–2 are the sample value.
24	`ModeN SOF Line`	SOF line register. The line upon which the SOF occurs measured from the rising edge of the F0 vertical sync. This is a 13-bit register.
25	`ModeN Vcoclk Divider`	Number of cycles of `vid_clk` (`vcoclk`) before `vcoclk_div` signal triggers. This is a 14-bit register.
26	`ModeN Ancillary Line`	The line to start inserting ancillary data packets.
27	`ModeN F0 Ancillary Line`	The line in field F0 to start inserting ancillary data packets.
28	`ModeN H-Sync Polarity`	Specify positive or negative polarity for the horizontal sync.

continued...

Address	Register	Description
		• Bit 0 is 0 for falling edge pulses (negative polarity). • Bit 0 is 1 for rising edge hsync pulses (positive polarity).
29	Mode*N* V-Sync Polarity	Specify positive or negative polarity for the vertical sync. • Bit 0 is 0 for falling edge pulses (negative polarity). • Bit 0 is 1 for rising edge vsync pulses (positive polarity).
30	Mode*N* Valid	Video mode valid. Set to indicate that this mode is valid and can be used for video output.

Note: To ensure the vid_f signal rises at the Field 0 blanking period and falls at the Field 1, use the following equations:

- F rising edge line ≥ Vertical blanking rising edge line
- F rising edge line < Vertical blanking rising edge line + (Vertical sync + Vertical front porch + Vertical back porch)
- F falling edge line < active picture line

8. 2D FIR II IP Core

The 2D FIR II IP core performs 2D convolution using matrices of specific coefficients.

The 2D FIR II IP core has 2 distinct modes of operation (user-defined at compile time).

- Standard FIR mode
  - In this mode, the IP core performs 2D finite impulse response filtering (convolution) using matrices of N×M coefficients, where N is a parameterizable number of horizontal taps (1 <= N <= 16) and M is a parameterizable number of vertical taps (1 <= M <= 16).
  - You can set the coefficients used by the filter either as fixed parameters at compile time, or as run-time alterable values which you can edit through an Avalon-MM slave interface.
  - With suitable coefficients, the filter can perform operations such as sharpening, smoothing, and edge detection.
- Dedicated edge-adaptive sharpening mode
  - You can use this mode to sharpen blurred edges in the incoming video.

8.1. FIR Filter Processing

The FIR II IP calculates the output pixel values in 3 stages.

1. Creates kernel

   The IP creates an *N*×*M* array of input pixels around the input pixel at the same position in the input image as the position of the output pixel in the output image. This center pixel has (*N*-1)/2 pixels to its left and *N*/2 pixels to its right in the array, and (*M*-1)/2 lines above it and *M*/2 lines below it.

   When the pixels to the left, right, above, or below the center pixel in the kernel extend beyond the edges of the image, the filter uses either replication of the edge pixel or full data mirroring, according to the value of a compile time parameter.

2. Convolutes

   The IP multiplies each pixel in the *N*×*M* input array by the corresponding coefficient in the *N*×*M* coefficient array. The IP sums the results to produce the filtered value.

   The FIR Filter II IP retains full precision throughout the calculation of each filtered value, with all rounding and saturation to the required output precision applied as a final stage.

3. Rounds and saturates.

   The IP rounds and saturates the resulting full precision filtered value according to the output precision specification

8.2. FIR Filter Precision

The FIR IP does not lose calculation precision during the FIR calculation.

- You may parameterize the input data to between 8 to 16 bits per color per pixel. The IP treats this data as unsigned integer data. You may enable optional guard bands at the input to keep the data inside a reduced range of values.
- You may parameterize the coefficient data up to a total width of 18 bits per coefficient. The coefficients may be signed or unsigned and contain up to 18 fractional bits.
- You may parameterize the output data to between 8 to 16 bits per color per pixel, and the selected output data width may be different from the input data width.

To convert from the full precision result of the filtering to the selected output precision, the IP first rounds up the value to remove the required number of fraction bits. Then the IP saturates the value. You may select how many fraction bits should be preserved in the final output using the IP parameter editor. As with the input data, the output data is treated as unsigned, so the IP clips any negative values that result from the filtering to 0. Any values greater than the maximum value that can be represented in the selected number of bits per color per pixel are clipped to this maximum value.

8.3. FIR Coefficient Specification

You can either specify the FIR filter IP filtering operation coefficients as fixed values that are not run-time editable, or you can turn on an Avalon memory-mapped agent interface to edit the values of the coefficients at run time.

The IP requires you to define a fixed-point type for the coefficients. The user-entered coefficients (shown as white boxes in the parameter editor) are rounded to fit in the chosen coefficient fixed-point type (shown as purple boxes in the parameter editor).

- For run-time editable coefficients, you must enter the desired coefficient values through an Avalon memory-mapped agent control interface at run time, and you can update the coefficient values as often as once per frame.

  Note: The coefficient values all revert to 0 after every reset, so you must initialize coefficients at least once on start-up.
- To keep the register map as small as possible and to reduce complexity in the hardware, the IP reduces the number of coefficients that it edits at run time when you turn on any of the symmetric modes.
- For T unique coefficient values after symmetry, the register map contains T addresses into which you should write coefficients, starting at address 7 and finishing at T+ 6.

Fixed Coefficient

For fixed coefficients, you specify the values for the coefficients with a comma-separated `.csv` text file and there is no Avalon memory-mapped agent interface. The selected coefficient values take effect immediate effect at reset.

Regardless of the symmetry mode, the text file must contain a full listing of all the coefficients in the *N*×*M* array i.e. the file must always contain *N*×*M* comma-separated values. When the `.csv` file is parsed in Platform Designer to create the list of compile time coefficients, the IP checks the values entered against the selected symmetry mode and provides warnings if the coefficients are not symmetric across the selected

axes. The values specified in the `.csv` file must be in their unquantized format. For example, if you want a value of 1.7 for a given coefficient, the value in the file should be 1.75. When the file is parsed in Platform Designer, the coefficients automatically quantize according to the precision you specify.

Note: The quantization process selects the closest value available in the given precision format. If you select the coefficients arbitrarily without reference to the available precision, the quantized value may differ from the desired value.

Run-time Editable Coefficients

To keep the register map as small as possible and to reduce complexity in the hardware, the number of coefficients that are edited at run time is reduced when you turn on any of the symmetric modes.

If the IP has T unique coefficient values after symmetry, the register map contains T addresses into which you should write coefficients, starting at address 7 and finishing at T+ 6.

Write coefficient index 0 (as described in the symmetry section) to address 7. Then write each successively indexed coefficient at each following address. The updated coefficient set does not take effect until you issue a write to address 6. You can write any value to address 6. The action of the write forces the commit.

The new coefficient set then takes effect on the next frame after the write to address 6.

Note: The coefficient values you write to the register map must be in prequantized format as the hardware cost to implement quantization on floating point values is prohibitive.

8.4. FIR Filter Symmetry

The FIR Filter IP supports symmetry modes for coefficient data.

The IP provides five symmetry modes for you to select:

- No symmetry
- Horizontal symmetry
- Vertical symmetry
- Horizontal and vertical symmetry
- Diagonal symmetry

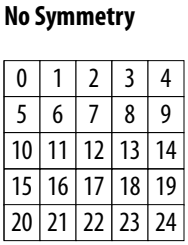
8.4.1. No Symmetry

The FIR Filter IP has no axes of symmetry in the 2D coefficient array.

The number of horizontal taps (*N*) and the number of vertical taps (*M*) may both be even or odd numbers.

With run-time control of the coefficient data, the register map includes *N*×*M* addresses to allow the value of each coefficient to be updated individually. The IP indexes the coefficients within the register map in raster scan order.

Figure 43. No Symmetry



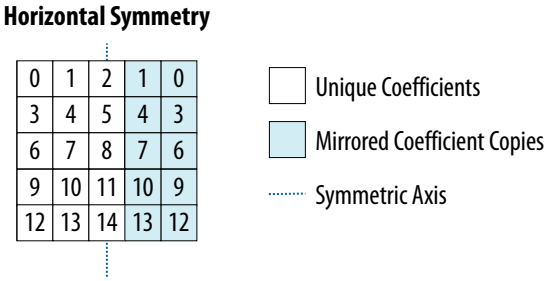
8.4.2. Horizontal Symmetry

The FIR Filter IP has 1 axis of symmetry across a vertical line through the center tap in the 2D coefficient array.

The number of vertical taps (*M*) may be even or odd, but the number of horizontal taps (*N*) must be even. Horizontal symmetry offers only ((*N*+1)/2)×*M* unique coefficient values. The remaining values in the array are mirrored duplicates.

With run-time control of the coefficients, the register map only includes addresses to update the ((*N*+1)/2)×*M* unique coefficient values, indexed for an example 5×5 array as shown in the figure below.

Figure 44. Horizontal Symmetry



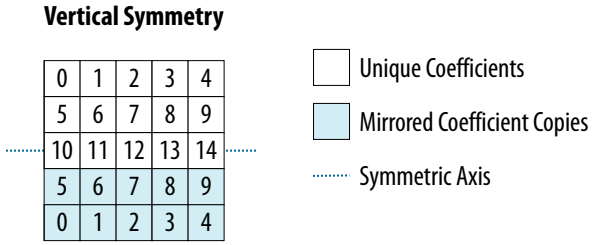
8.4.3. Vertical Symmetry

The FIR Filter IP has 1 axis of symmetry across a horizontal line through the center tap in the 2D coefficient array.

In this case, the number of horizontal taps (*N*) may be even or odd, but the number of vertical taps (*M*) must be even. Vertical symmetry offers only *N*×((*M*+1)/2) unique coefficient values, with the remaining values in the array being mirrored duplicates.

With run-time control of the coefficients, the register map only includes addresses to update the *N*×((*M*+1)/2) unique coefficient values, indexed for an example 5×5 array.

Figure 45. Vertical Symmetry



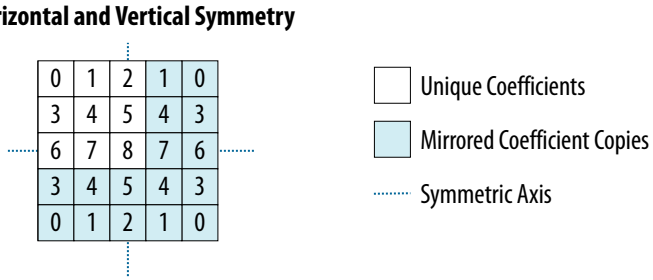
8.4.4. Horizontal and Vertical Symmetry

The FIR Filter IP has 2 axes of symmetry across a horizontal line and a vertical line through the center tap in the 2D coefficient array.

In this case, the number of horizontal taps (*N*) and the number of vertical taps (*M*) must be even. Horizontal and vertical symmetry offers only ((*N*+1)/2)×((*M*+1)/2) unique coefficient values. The remaining values in the array are mirrored duplicates.

With run-time control of the coefficients, the register map only includes addresses to update the ((*N*+1)/2)×((*M*+1)/2) unique coefficient values, indexed for an example 5×5 array.

Figure 46. Horizontal and Vertical Symmetry



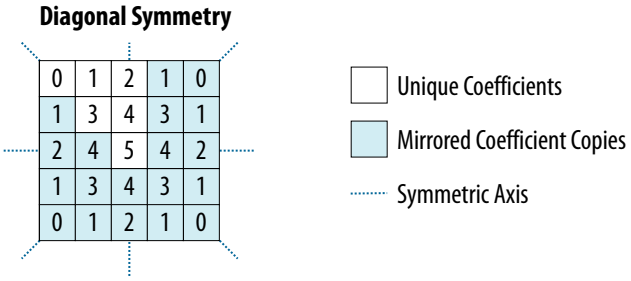
8.4.5. Diagonal Symmetry

The FIR Filter IP has 4 axes of symmetry in the 2D coefficient array across a horizontal line, a vertical line, and 2 diagonal lines through the center tap.

In this case, the number of horizontal taps (*N*) and the number of vertical taps (*M*) must be even, and they must have same value (*N* = *M*). Diagonal symmetry offers only Tu = ((*N*+1)/2) unique coefficient values in either the horizontal or vertical directions, and a total of (Tu*(Tu+1))/2 unique coefficient values.

With run-time control of the coefficients, the register map only includes addresses to update the (Tu *(Tu +1))/2 unique coefficient values, indexed for an example 5×5 array.

Figure 47. Diagonal Symmetry



8.5. Result to Output Data Type Conversion

After calculation, the FIR Filter IP converts the fixed-point type of the results to the integer data type of the output.

1. Scales result. Scaling quickly increases the color depth of the output. You can shift the binary point right –16 to +16 places. The IP implements scaling as a simple shift operation so it does not require multipliers.
2. Removes fractional bits. If any fractional bits exist, you can choose to remove them through these methods:
   - Truncate to integer. The IP removes fractional bits from the data; equivalent to rounding towards negative infinity.
   - Round half up. The IP rounds up to the nearest integer. If the fractional bits equal 0.5, rounding is towards positive infinity.
   - Round half even. The IP rounds to the nearest integer. If the fractional bits equal 0.5, rounding is towards the nearest even integer.
3. Convert from signed to unsigned. If any negative numbers exist in the results and the output type is unsigned, you can convert to unsigned through these methods:
   - Saturate to the minimum output value (constraining to range).
   - Replace negative numbers with their absolute positive value.
4. Constrain to range. If any of the results are beyond a specific range, the IP automatically adds logic to saturate the results to the minimum and maximum output values. The specific range is the specified range of the output guard bands, or if unspecified, the minimum and maximum values allowed by the output bits per pixel.

8.6. Edge-Adaptive Sharpen Mode

In edge-adaptive sharpen mode, the 2D FIR II IP core attempts to detect blurred edges and applies a sharpening filter only to the pixels that are determined to be part of a blurred edge.

8.6.1. Edge Detection

You can determine what constitutes a blurred edge by setting upper and lower blur thresholds (either at compile time or as run-time controlled values). The edge detection is applied independently in the horizontal and vertical directions according to the following steps:

1. The target pixel is compared to the pixels immediately to the left and right.
2. If both differences to the neighboring pixel are less than the upper blur threshold, the edge detection continues. Otherwise, the pixel is considered to be part of an edge that is already sharp enough that does not require further sharpening.
3. If the differences to the pixels to the left and right are below the upper blur threshold, the differences between the target pixel and its neighbors 1, 2, and 3 pixels to the left and right are calculated.
4. You may configure the range of pixels over which a blurred edge is detected.
   - Setting the `Blur search range` register to 1 means that only the differences to the neighbors 1 pixel to the left and right are considered.
   - Setting the register to 2 increases the search across the 2 neighboring pixels.
   - Setting the register to 3 increases it to the full range across all 3 neighbors in each direction.

   The value of the blur search range can be updated at run time if you turn on run-time control feature for the 2D FIR II IP core.
5. If the difference to any of the enabled neighboring pixels is greater than the lower blur threshold, the target pixel is tagged as a horizontal edge. Otherwise, target pixel is considered to be part of an area of flat color and left unaltered.

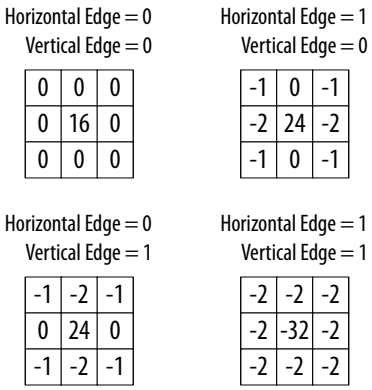
Note: The algorithm is described for horizontal edge detection, but the same algorithm is used for vertical detection, just replace *left* and *right* with *above* and *below.*

8.6.2. Filtering

Depending on whether each pixel is tagged as a horizontal and/or vertical edge, 1 of 4 different 3×3 filter kernels is applied (and the result divided by 16) to produce the final output pixel.

Note: In edge-adaptive sharpen, the filter is fixed at 3×3 taps and all parameters regarding coefficient precision are ignored. There is also no option to override the coefficients used in the filter kernels.

Figure 48. Edge-adaptive Sharpen Filtering



8.6.3. Precision

The edge-adaptive sharpen mode of the 2D FIR II IP core does not allow for a different setting in the bits per pixel per color plane.

- The input bits per color plane is maintained at the output.
- The output of the filtering kernel is rounded to the nearest integer (by adding 8 prior to the divide by 16).
- Any negative values are clipped to 0 and any values greater than the maximum value that can be represented in the selected number of bits per color per pixel are clipped to this maximum value.

8.7. 2D FIR Filter Parameter Settings

Table 30. 2D FIR II Parameter Settings

Parameter	Value	Description
Number of color planes	1, 2, **3**, 4	Select the number of color planes per pixel.
Color planes transmitted in parallel	**On** or Off	Select whether to send the color planes in parallel or in sequence (serially).
Number of pixels in parallel	**1**, 2, 4, 8	Select the number of pixels transmitted per clock cycle.
4:2:2 video data	On or **Off**	Turn on if the input data is 4:2:2 formatted, otherwise the data is assumed to be 4:4:4 formatted. *Note:* The IP core does not support odd heights or widths in 4:2:2 mode.
Maximum frame width	32-8192, Default = **1920**	Specify the maximum frame width allowed by the IP core.
Maximum frame height	32-8192, Default = **1080**	Specify the maximum frame height allowed by the IP core.
Input bits per pixel per color plane	4-20, Default = **8**	Select the number of bits per color plane per pixel at the input.
Enable input guard bands	On or **Off**	Turn on to limit the range for each input color plane.
Lower input guard bands	0– $2^{(\text{input bits per symbol})-1}$ Default = **0**	Set the lower range limit to each input color plane. Values beneath this will be clipped to this limit.
Upper input guard bands	0– $2^{(\text{input bits per symbol})-1}$ Default = **255**	Set the upper range limit to each input color plane. Values above this will be clipped to this limit.

continued...

Parameter	Value	Description
Output bits per pixel per color plane	4–20, Default = **8**	Select the number of bits per color plane per pixel at the output.
Enable output guard bands	On or **Off**	Turn on to limit the range for each output color plane.
Lower output guard bands	0– $2^{(\text{input bits per symbol})-1}$ Default = **0**	Set the lower range limit to each output color plane. Values beneath this will be clipped to this limit.
Upper output guard bands	0– $2^{(\text{input bits per symbol})-1}$ Default = **255**	Set the upper range limit to each output color plane. Values above this will be clipped to this limit.
Filtering algorithm	• **STANDARD_FIR** • EDGE_ADAPTIVE_SHARPEN	Select the preferred FIR mode.
Enable edge data mirroring	**On** or Off	Turn on to enable full mirroring of data at frame/field edges. If you do not turn on this feature, the edge pixel will be duplicated to fill all filter taps that stray beyond the edge of the frame/field.
Vertical filter taps	1–16, Default = **8**	Select the number of vertical filter taps.
Horizontal filter taps	1–16, Default = **8**	Select the number of horizontal filter taps.
Vertically symmetric coefficients	On or **Off**	Turn on to specify vertically symmetric coefficients.
Horizontally symmetric coefficients	On or **Off**	Turn on to specify horizontally symmetric coefficients.
Diagonally symmetric coefficients	On or **Off**	Turn on to specify diagonally symmetric coefficients.
Blur search range	1–3, Default = **1**	Select the number of pixels over which a blurred edge may be detected. This option is available only if you select **EDGE_ADAPTIVE_SHARPEN**. If you enable **Run-time control**, you may override this value using the Avalon-MM control slave interface at run time.
Rounding method	• TRUNCATE • **ROUND_HALF_UP** • ROUND_HALF_EVEN	Select how fraction bits are treated during rounding. • TRUNCATE simply removes the unnecessary fraction bits. • ROUND_HALF_UP rounds to the nearest integer value, with 0.5 always being rounded up. • ROUND_HALF_EVEN rounds 0.5 to the nearest even integer value.
Use signed coefficients	**On** or Off	Turn on to use signed coefficient values.
Coefficient integer bits	0–16, Default = **1**	Select the number of integer bits for each coefficient value.
Coefficient fraction bits	0–24, Default = **7**	Select the number of fraction bits for each coefficient value.
Move binary point right	–16 to +16, Default = **0**	Specify the number of places to move the binary point to the right prior to rounding and saturation. A negative value indicates a shift to the left.
Run-time control	On or **Off**	Turn on to enable coefficient values to be updated at run-time through the Avalon-MM control slave interface. *Note:* When you turn on this parameter, the `Go` bit gets deasserted by default. When you turn off this parameter, the `Go` is asserted by default.
Fixed coefficients file (Unused if run-time updates of coefficients is enabled.)	User specified file (including full path to locate the file)	If you do not enable run-time control, you must specify a CSV containing a list of the fixed coefficient values.

continued...

Parameter	Value	Description
Default upper blur limit (per color plane)	0– $2^{(\text{input bits per symbol})-1}$ Default = **0**	Sets the default upper blur threshold for blurred edge detection. This option is available only if you select **EDGE_ADAPTIVE_SHARPEN**. If you enable **Run-time control**, you may override this value using the Avalon-MM control slave interface at run time.
Default lower blur limit (per color plane)	0– $2^{(\text{input bits per symbol})-1}$ Default = **0**	Sets the default lower blur threshold for blurred edge detection. This option is available only if you select **EDGE_ADAPTIVE_SHARPEN**. If you enable **Run-time control**, you may override this value using the Avalon-MM control slave interface at run time.
Reduced control register readback	On or **Off**	If you turn on this parameter, the values written to register 3 and upwards cannot be read back through the control slave interface. This option reduces ALM usage. If you do not turn on this parameter, the values of all the registers in the control slave interface can be read back after they are written.
How user packets are handled	• No user packets allowed • **Discard all user packets received** • Pass all user packets through to the output	• If your design does not require the 2D FIR II IP core to propagate user packets, then you may select **Discard all user packets received** to reduce ALM usage. • If your design guarantees there will never be any user packets in the input data stream, then you can further reduce ALM usage by selecting **No user packets allowed**. In this case, the 2D FIR II IP core may lock if it encounters a user packet.
Add extra pipelining registers	On or **Off**	Turn on to add extra pipeline stage registers to the data path. You must to turn on this option to achieve: • Frequency of 150 MHz for Cyclone V devices • Frequencies above 250 MHz for Arria V, Stratix V, or Intel Arria 10 devices
Video no blanking	On or **Off**	Turn on if the input video does not contain vertical blanking at its point of conversion to the Avalon-ST video protocol.

8.8. 2D FIR Filter Control Registers

Table 31. 2D FIR Filter II Control Register Map

The 2D FIR Filter II IP core filtering operation coefficients can either be specified as fixed values that are not run-time editable, or you can opt to enable an Avalon-MM slave interface to edit the values of the coefficients at run time. When a control slave interface is included, the IP core resets into a stopped state and must be started by writing a '1' to the Go bit of the control register before any input data is processed.

Address	Register	Description
0	`Control`	Bit 0 of this register is the Go bit, all other bits are unused. Setting this bit to 0 causes the IP core to stop at the end of the next frame/field packet. When you enable run-time control, the Go bit gets deasserted by default. If you do not enable run-time control, the Go is asserted by default.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused. The IP core sets this address to 0 between frames. The IP core sets this address to 1 when it is processing data and cannot be stopped.
2	`Interrupt`	This bit cannot be used because the IP core does not generate any interrupts.

continued...

Address	Register	Description
3	`Blur search range`	Set this register to 1, 2, or 3 to override the default parameter setting for the edge detection range in edge-adaptive sharpen mode.
4	`Lower blur threshold`	This register updates the value of the lower blur threshold used in edge detection in edge-adaptive sharpen mode.
5	`Upper blur threshold`	This register updates the value of the upper blur threshold used in edge detection in edge-adaptive sharpen mode.
6	`Coefficient commit`	Writing any value to this register causes the coefficients currently in addresses 7 to (6+T) to be applied from the start of the next input.
7–(6+ T)	`Coefficient data`	Depending on the number of vertical taps, horizontal taps, and symmetry mode, T addresses are allocated to upload the T unique coefficient values required to fill the 2D coefficient array.

9. Mixer II IP Core

The Mixer II IP core mixes together multiple image layers.

The run-time control is partly provided by an Avalon-MM slave port with registers for the location, and on or off status of each foreground layer. The dimensions of each layer are then specified by Avalon-ST Video control packets.

- Each foreground layer must be driven by a frame buffer or frame reader so that data can be provided at the correct time.
  - If the **Synchronize background to layer 0** parameter is enabled, the input connected to layer 0 is used as the background layer and does not have to be driven by a frame buffer or frame reader.
  - If the **Synchronize background to layer 0** parameter is disabled, all layers are treated as foreground layers and must be driven by a frame buffer or frame reader.
- Each layer must fit within the dimensions of the background layer.

  To display the layers correctly:
  - The rightmost edge of each layer (width + X offset) must fit within the dimensions of the background layer.
  - The bottom edge of each layer (height + Y offset) must fit within the dimensions of the background layer.

Note: If these conditions are not met for any layers, the Mixer II IP core will not display those layers. However, the corresponding inputs will be consumed and will not get displayed.

The Mixer II IP core has the following features:

- Supports picture-in-picture mixing and image blending with per-pixel and static value alpha support.
- Supports dynamic changing of location and size of each layer during run time.
- Supports dynamic changing of layers positioning during run time.
- Allows the individual layers to be switched on and off.
- Supports up to 8 pixels in parallel.
- Supports up to 20 inputs.
- Includes built in test pattern generator as background layer if required.
- Supports low-latency mode by using layer 0 as background layer.

The Mixer II IP core reads the control data in two steps at the start of each frame. The buffering happens inside the IP core so that the control data can be updated during the frame processing without unexpected side effects.

The first step occurs after the IP core processes and transmits all the non-image data packets of the background layer, and it has received the header of an image data packet of type 0 for the background. At this stage, the on/off status of each layer is read. A layer can be:

- disabled (0),
- active and displayed (1), or
- consumed but not displayed (2)

The maximum number of image layers mixed cannot be changed dynamically and must be set in the parameter editor.

The IP core processes the non-image data packets of each active foreground layer, displayed or consumed, in a sequential order, layer 1 first. The IP core integrally transmits the non-image data packets from the background layer. The IP core treats the non-image data packets from the foreground layers differently depending on their type.

- Control packets (type 15)— processed to extract the width and height of each layer and are discarded on the fly.
- Other/user packets (types 1–14)—propagated unchanged.

The second step corresponds to the usual behavior of other Video and Image Processing IP cores that have an Avalon-MM slave interface. After the IP core has processed and/or propagated the non-image data packets from the background layer and the foreground layers, it waits for the Go bit to be set to 1 before reading the top left position of each layer.

Consequently, the behavior of the Mixer II IP core differs slightly from the other Video and Image Processing IP cores, as illustrated by the following pseudo-code:

```
go = 0;
while (true)
{
    status = 0;
    read_non_image_data_packet_from background_layer();
    read_control_first_pass(); // Check layer status
                                          (disable/displayed/consumed)
    for_each_layer layer_id
    {
        // process non-image data packets for displayed or consumed
                                                              layers
        if (layer_id is not disabled)
        {

        handle_non_image_packet_from_foreground_layer(layer_id);
        }
    }
    while (go != 1)
        wait;
    status = 1;
    read_control_second_pass(); // Copies top-left coordinates to
                                                          internal registers
    send_image_data_header();
    process_frame();
}
```

9.1. Alpha Blending

When you turn on **Alpha Blending Enable** in the Mixer II parameter editor, the IP core is able to mix layers with varying levels of translucency using an alpha value obtained from a run-time programmable register setting.

Each input (*N*) has an **Input*N* alpha channel** parameter that enables the extra per-pixel alpha value for input *n*. When you enable either of the Alpha blending modes, bit [3:2] in the `Input control` *N* registers control which one of these alpha values is used:

- Fixed opaque alpha value
- Static (run-time programmable) value (only when you select the **Alpha Blending Enable** parameter)
- Per-pixel streaming value (only when you select the **Input*N* alpha channel** parameter for Input*N*)

Note: When you turn the **Input*N* alpha channel** parameter, the IP core adds an extra symbol per-pixel for that input. The least significant symbol is the alpha value. The control packet is composed of all symbols including alpha.

The valid range of alpha coefficients is 0 to 1, where 1 represents full translucence, and 0 represents fully opaque.

The Mixer II IP core determines the alpha value width based on your specification of the bits per pixel per color parameter. For *n*-bit alpha values, the coefficients range from 0 to 2^n–1. The model interprets (2^n– 1) as 1, and all other values as (Alpha value/2^n. For example, 8-bit alpha value 255>1, 254>254/256, 253>253/256, and so on.

The value of an output pixel O_N, where *N* ranges from 1 to number of inputs minus 1, is derived from the following recursive formula:

$$O_N = (1 - a_N)\ p_N + a_N\ O_{N-1}$$

$$O_0 = (1 - a_0)\ p_0 + a_0\ p_{background}\ ,$$

where p_N is the input pixel for layer *N*, a_N is the alpha value for layer *N*, and $p_{background}$ is the background pixel value.

The Mixer II IP core skips consumed and disabled layers.

Note: All input data samples must be in unsigned format. If the number of bits per pixel per color plane is *N*, then each sample consists of *N* bits of data, which are interpreted as an unsigned binary number in the range [0,2^n–1]. The Mixer II IP core also transmits all output data samples in the same unsigned format.

9.2. Mixer II Parameter Settings

Table 32. Mixer II Parameter Settings

Parameter	Value	Description
Maximum output frame width	32-8192, Default = **1920**	Specify the maximum image width for the layer background in pixels.
Maximum output frame height	32-8192, Default = **1080**	Specify the maximum image height for the layer background in pixels.
Bits per color sample	4-20, Default = **8**	Select the number of bits per pixel (per color plane).
Number of pixels in parallel	**1**, 2, 4, 8	Select the number of pixels transmitted every clock cycle.
Colorspace (used for background layer)	• **RGB** • YCbCr	Select the color space you want to use for the background test pattern layer.
4:2:2 support	On or **Off**	Turn on to enable 4:2:2 sampling rate format for the background test pattern layer. Turn off to enable 4:4:4 sampling rate. *Note:* The IP core does not support odd heights or widths in 4:2:2 mode.
Synchronize background to layer 0	On or **Off**	Turn on to synchronize the background layer to layer 0 (low-latency mode). Turn off to use the internal Test Pattern Generator as the background layer.
Layer Position Enable	On or **Off**	Turn on to enable the layer mapping. Turn off to disable the layer mapping functionality to save gates.
Number of inputs	1-20; Default = **4**	Specify the number of inputs to be mixed.
Alpha Blending Enable (all layers)	On or **Off**	Turn on to allow the IP core to alpha blend using a run-time programmable register alpha value..
Input*N* alpha channel	On or **Off**	Turn on to allow the input streams to have an alpha channel.
Pattern	• **Color bars** • Uniform background	Select the pattern you want to use for the background test pattern layer.
R or Y	Default = 0	If you choose to use uniform background pattern, specify the individual R'G'B' or Y'Cb'Cr' values based on the color space you selected. The uniform values match the width of bits per pixel up to a maximum of 16 bits. Values beyond 16 bits are zero padded at the LSBs.
G or Cb	Default = 0	
B or Cr	Default = 0	
Register Avalon-ST ready signals	On or **Off**	Turn on to add pipeline. Adding pipeline increases the f_{MAX} value when required, at the expense of increased resource usage.
Reduced control register readback	On or **Off**	If you turn on this parameter, the values written to register 3 and upwards cannot be read back through the control slave interface. This option reduces ALM usage. If you do not turn on this parameter, the values of all the registers in the control slave interface can be read back after they are written.
Add extra register stages to data pipeline	1-20; Default = **0**	Add extra register stages in the data pipeline to improve timing.

continued...

Parameter	Value	Description
		Add one register stage every *N*th data processing stage where *N* is the parameter value; setting a value of 0 disables this feature. There are as many data processing stages as there are inputs.
How user packets are handled	• No user packets allowed • **Discard all user packets received** • Pass all user packets through the output	Select whether to allow user packets to be passed through the mixer.

9.3. Mixer II Control Registers

For efficiency reasons, the Video and Image Processing Suite IP cores buffer a few samples from the input stream even if they are not immediately processed. This implies that the Avalon-ST inputs for foreground layers assert ready high, and buffer a few samples even if the corresponding layer has been deactivated.

Table 33. Mixer II Control Register Map

The table describes the control register map for Mixer II IP core.

Address	Register	Description
0	`Control`	Bit 0 of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the IP core to stop the next time control information is read.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused.
2	`Reserved`	Reserved for future use.
3	`Background Width`	Change the width of the background layer for the next and all future frames. Not available when the **Synchronize background to layer 0** parameter is enabled.
4	`Background Height`	Changes the height of the background layer for the next and all future frames. Not available when the **Synchronize background to layer 0** parameter is enabled.
5	`Uniform background Red/Y`	Specifies the value for R (RGB) or Y (YCbCr). If you choose to use uniform background pattern, specify the individual R'G'B' or Y'Cb'Cr' values based on the color space you selected. The uniform values match the width of bits per pixel up to a maximum of 16 bits. The IP core zero-pads values beyond 16 bits at the LSBs.
6	`Uniform background Green/Cb`	Specifies the value for G (RGB) or Cb (YCbCr). If you choose to use uniform background pattern, specify the individual R'G'B' or Y'Cb'Cr' values based on the color space you selected. The uniform values match the width of bits per pixel up to a maximum of 16 bits. The IP core zero-pads values beyond 16 bits at the LSBs.
7	`Uniform background Blue/Cr`	Specifies the value for B (RGB) or Cr (YCbCr). If you choose to use uniform background pattern, specify the individual R'G'B' or Y'Cb'Cr' values based on the color space you selected. The uniform values match the width of bits per pixel up to a maximum of 16 bits. The IP core zero-pads values beyond 16 bits at the LSBs.
8+5*n*	`Input X offset` *n*	X offset in pixels from the left edge of the background layer to the left edge of input *n*.

continued...

Address	Register	Description
		Note: n represents the input number, for example input 0, input 1, and so on.
9+5*n*	`Input Y offset n`	Y offset in pixels from the top edge of the background layer to the top edge of input *n*. *Note: n* represents the input number, for example input 0, input 1, and so on.
10+5*n*	`Input control n`	• Set to bit 0 to enable input *n*. • Set to bit 1 to enable consume mode. • Set to bits 3:2 to enable alpha mode. — 00 – No blending, opaque overlay — 01 – Use static alpha value (available only when you turn on the **Alpha Blending Enable** parameter.) — 10 – Use alpha value from input stream (available only when you turn on the **Input*N* alpha channel** parameter.) — 11 – Unused *Note: n* represents the input number, for example input 0, input 1, and so on.
11+5*n*	`Layer position n`	Specifies the layer mapping functionality for input *n*. Available only when you turn on the **Layer Position Enable** parameter. *Note: n* represents the input number, for example input 0, input 1, and so on.
12+5*n*	`Static alpha n`	Specifies the static alpha value for input *n* with bit width matching the **bits per pixel per color plane** parameter. Available only when you turn on the **Alpha Blending Enable** parameter. *Note: n* represents the input number, for example input 0, input 1, and so on.

9.4. Layer Mapping

When you turn on **Layer Position Enable** in the Mixer II parameter editor, the Mixer II allows a layer mapping to be defined for each input using the Layer Position control registers.

The layer positions determine whether an input is mixed in the background (layer 0) through to the foreground (layer *N*, where *N* is the number of inputs minus one) in the final output image.

Note: if there are any repeated values within the Layer Position registers (indicating that two inputs are mapped to the same layer), the input with the repeated layer position value will not be displayed and will be consumed.

If you turn off the **Layer Position Enable** parameter, the Mixer II IP core uses a direct mapping between the ordering of the inputs and the mixing layers. For example, Layer 0 will be mapped to Input 0, Layer 1 to Input 1, and so on.

9.5. Low-Latency Mode

When you turn on **Synchronize background to layer 0** in the Mixer II parameter editor, the IP core operates in low-latency mode.

In low-latency mode, the Mixer II IP core synchronizes its operations to the incoming video on layer 0. Each video frame received on layer 0 generates a frame at the IP core's output. The dimensions of the output video matches the dimensions of the input frame.

For layer mapping in low-latency mode, the Mixer II IP core functions in the following behavior:

- If you turn off the **Layer Position Enable** parameter when the IP core is in low-latency mode, the Mixer II IP core synchronizes its operation to input 0 (for this parameterization, input 0 will be mapped to layer 0).
- If you turn on the **Layer Position Enable** parameter when the IP core is in low-latency mode, the Mixer IP core synchronizes its operation to the input that is mapped to layer 0 by its `Layer position` *n* control register.

10. Clipper II IP Core

The Clipper II IP core provides a means to select an active area from a video stream and discard the remainder.

You can specify the active region by providing the offsets from each border or a point to be the top-left corner of the active region along with the region's width and height.

The Clipper II IP core handles changing input resolutions by reading Avalon-ST Video control packets. An optional Avalon-MM interface allows the clipping settings to be changed at run time.

10.1. Clipper II Parameter Settings

Table 34. Clipper II Parameter Settings

Parameter	Value	Description
Maximum input frame width	32–8192, Default = **1920**	Specify the maximum frame width of the clipping rectangle for the input field (progressive or interlaced).
Maximum input frame height	32–8192, Default = **1080**	Specify the maximum height of the clipping rectangle for the input field (progressive or interlaced).
Bits per pixel per color sample	4–20, Default = **8**	Select the number of bits per color plane.
Number of color planes	1–4, Default = **2**	Select the number of color planes per pixel.
Number of pixels in parallel	**1**, 2, 4, 8	Select the number of pixels in parallel.
Color planes transmitted in parallel	**On** or Off	Select whether to send the color planes in parallel or serial. If you turn on this parameter, and set the number of color planes to 3, the IP core sends the R'G'B's with every beat of data.
Run-time control	**On** or Off	Turn on if you want to specify clipping offsets using the Avalon-MM interface. *Note:* When you turn on this parameter, the Go bit gets deasserted by default. When you turn off this parameter, the Go is asserted by default.
Clipping method	• **OFFSETS** • RECTANGLE	Specify the clipping area as offsets from the edge of the input area or as a fixed rectangle.
Left offset	0–8192, Default = **0**	Specify the *x* coordinate for the left edge of the clipping rectangle. 0 is the left edge of the input area. *Note:* The left and right offset values must be less than or equal to the input image width.
Top offset	0–8192, Default = **0**	Specify the *y* coordinate for the top edge of the clipping rectangle. 0 is the top edge of the input area. *Note:* The top and bottom offset values must be less than or equal to the input image height.

continued...

Parameter	Value	Description
Right offset	0–8192, Default = **0**	Specify the *x* coordinate for the right edge of the clipping rectangle. 0 is the right edge of the input area. *Note:* The left and right offset values must be less than or equal to the input image width.
Bottom offset	0–8192, Default = **0**	Specify the *y* coordinate for the bottom edge of the clipping rectangle. 0 is the bottom edge of the input area. *Note:* The top and bottom offset values must be less than or equal to the input image height.
Width	32–8192, Default = **32**	Specify the width of the clipping rectangle. The minimum output width is 32 pixels.
Height	32–8192, Default = **32**	Specify the height of the clipping rectangle. The minimum output height is 32 pixels.
Add extra pipelining registers	On or **Off**	Turn on this parameter to add extra pipeline stage registers to the data path. You must turn on this parameter to achieve: • Frequency of 150 MHz for Cyclone III or Cyclone IV devices • Frequencies above 250 MHz for Arria II, Stratix IV, or Stratix V devices

10.2. Clipper II Control Registers

Table 35. Clipper II Control Register Map

The control data is read once at the start of each frame and is buffered inside the Clipper II IP core, so the registers can be safely updated during the processing of a frame.

Address	Register	Description
0	`Control`	Bit 0 of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the IP core to stop the next time control information is read. When you enable run-time control, the `Go` bit gets deasserted by default. If you do not enable run-time control, the `Go` is asserted by default.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused. The Clipper IP core sets this address to 0 between frames. It is set to 1 while the IP core is processing data and cannot be stopped.
2	`Interrupt`	This bit is not used because the IP core does not generate any interrupts.
3	`Left Offset`	The left offset, in pixels, of the clipping window/rectangle. *Note:* The left and right offset values must be less than or equal to the input image width.
4	`Right Offset or Width`	In clipping window mode, the right offset of the window. In clipping rectangle mode, the width of the rectangle. *Note:* The left and right offset values must be less than or equal to the input image width.
5	`Top Offset`	The top offset, in pixels, of the clipping window/rectangle. *Note:* The top and bottom offset values must be less than or equal to the input image height.
6	`Bottom Offset or Height`	In clipping window mode, the bottom offset of the window. In clipping rectangle mode, the height of the rectangle. *Note:* The top and bottom offset values must be less than or equal to the input image height.

11. Color Plane Sequencer II IP Core

The Color Plane Sequencer II IP core changes how color plane samples are transmitted across the Avalon-ST interface and rearranges the color patterns used to transmit Avalon-ST Video data packets.

A color pattern is a matrix that defines a pattern of color samples repeating over the length of an image.

The Color Plane Sequencer II IP core offers the following features:

- Splits or duplicates a single Avalon-ST Video stream into two or, conversely, combines two input streams into a single stream.
- Supports Avalon-ST Video streams with up to 4 pixels transmitted in parallel. A pixel may contain up to 4 color planes transmitted either in parallel or in sequence but not both
- The input/output color patterns to rearrange the Avalon-ST Video streams between the inputs and outputs may be defined over two pixels, which covers all common use-cases.

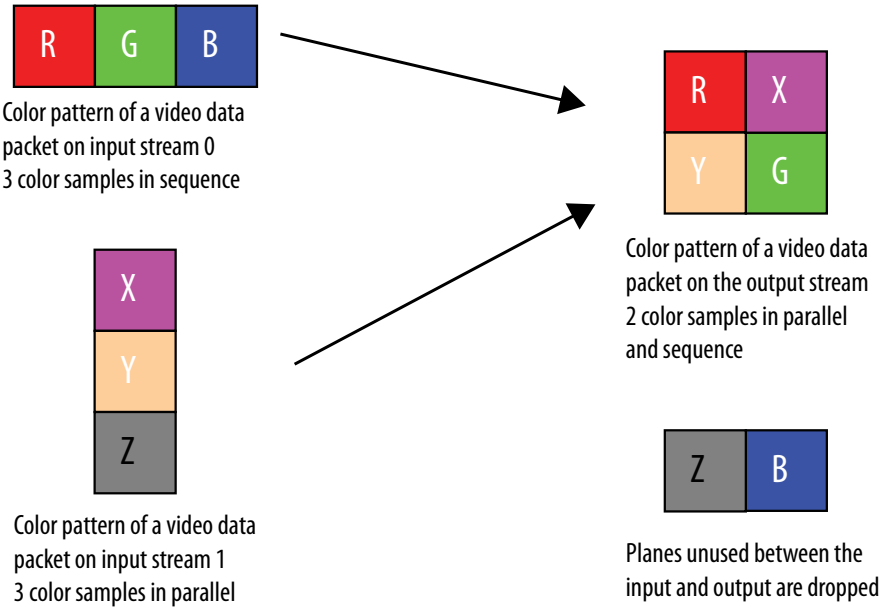
11.1. Combining Color Patterns

You can configure the Color Plane Sequencer II IP core to combine two Avalon-ST Video streams into a single stream.

In this mode of operation, the IP core pulls in two input color patterns (one for each input stream) and arranges to the output stream color pattern in a user-defined way, in sequence or parallel, as long as it contains a valid combination of the input channels. In addition to this combination and rearrangement, color planes can also be dropped.

Figure 49. Example of Combining Color Patterns

The figure shows an example of combining and rearranging two color patterns.

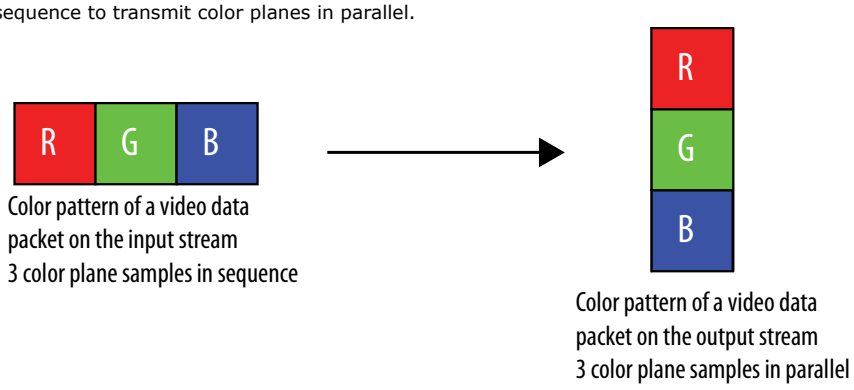


11.2. Rearranging Color Patterns

You can configure the Color Plane Sequencer II IP core to rearrange the color pattern of a video packet, and drop or duplicate color planes.

Figure 50. Example of Rearranging Color Patterns

The figure shows an example that rearranges the color pattern of a video data packet which transmits color planes in sequence to transmit color planes in parallel.



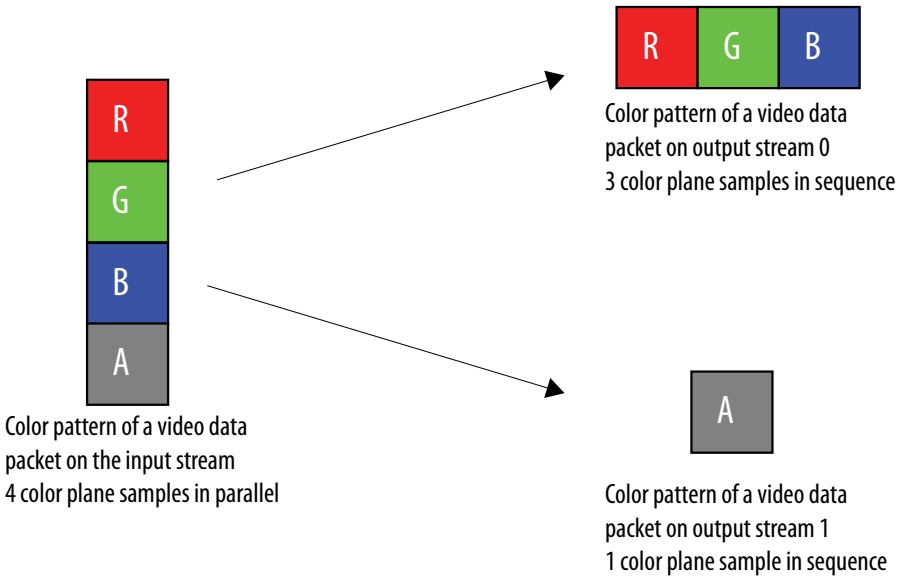
11.3. Splitting and Duplicating

You can configure the Color Plane Sequencer II IP core to split a single Avalon-ST Video input stream into two Avalon-ST Video output streams.

In this mode of operation, the IP core arranges the color patterns of video data packets on the output streams in a user-defined way using any of the color planes of the input color pattern.

The color planes of the input color pattern are available for use on either, both, or neither of the outputs. This allows for splitting of video data packets, duplication of video data packets, or a mix of splitting and duplication. The output color patterns are independent of each other, so the arrangement of one output stream's color pattern places no limitation on the arrangement of the other output stream's color pattern.

Figure 51. Example of Splitting Color Patterns



Caution: A deadlock may happen when a video design splits, processes independently, and then joins back the color planes, or when the sequencer splits the color planes in front of another VIP IP core. To avoid this issue, add small FIFO buffers at the output of the Color Plane Sequencer II IP core that are configured as splitters.

11.4. Handling of Subsampled Data

Besides fully sampled color patterns, the Color Plane Sequencer II IP core also supports 4:2:2 subsampled data.

To support 4:2:2 subsampled data, you must configure the IP core to use a 2-pixel pattern for the relevant input or output.

- When specifying an input pattern over two pixels, the Color Plane Sequencer II IP core pulls two input pixels from the corresponding input before doing the rearrangement. Hence, you can configure the first pixel of the pattern with color planes "Y" and "Cb" and the second pixel of the pattern with color planes "Y" and "Cr".
- When specifying an output pattern over two pixels, each rearrangement operation produces two output pixels. You may specify different color planes for the first and second pixel of the pattern.

You may use two-pixel patterns irrespective of the Avalon-ST Video transmission settings. They remain valid when pixels are transmitted in parallel or when color planes are transmitted sequentially.

The width of Avalon-ST Video control packets is automatically modified when handling subsampled data.

- When using a 2-pixel pattern for input stream 0, the IP core halves the width of the input control packets if the output is using a single-pixel pattern.
- When using a single pixel pattern for input stream 0, the IP doubles the width of the input control packets if the output is using a 2-pixel pattern.
- Control packet widths are not modified when using a single-pixel or a 2-pixel pattern on both sides.

11.5. Handling of Non-Image Avalon-ST Packets

The Color Plane Sequencer II IP core also handles Avalon-ST Video packets other than video data packets to the output(s).

You can forward Avalon-ST Video packets other than video data packets to the output(s) with these options:

- Avalon-ST Video control packets from input stream 1 are always dropped.
- Avalon-ST Video control packets from input stream 0 may be either propagated or dropped depending on the IP parameterization but the last control packet received before the image packet on input stream 0 is always propagated on all enabled outputs and its width may be altered.
- Input user packets can be dropped or forwarded to either or both outputs.

Note: When the color pattern of a video data packet changes from the input to the output side of a block, the IP core may pad the end of non-video user packets with extra data. Intel recommends that when you define a packet type where the length is variable and meaningful, you send the length at the start of the packet. User data is never truncated but there is no guarantee that the packet length will be preserved or even rounded up to the nearest number of output color planes.

11.6. Color Plane Sequencer Parameter Settings

Table 36. Color Plane Sequencer II Parameter Settings

n refers to the input or output number.

Parameter	Value	Description
How user packets are handled	• No user packets allowed • Discard all user packets received • **Pass all user packets through to the output(s)**	• If your design does not require the IP core to propagate user packets, then you may select **Discard all user packets received** to reduce ALM usage. • If your design guarantees there will never be any user packets in the input data stream, then you can further reduce ALM usage by selecting **No user packets allowed**. In this case, the IP core may lock if it encounters a user packet. • When propagating user packets, you should specify how the packets are routed. Each input can be routed to either or both outputs independently.
Add extra pipelining registers	On or **Off**	Turn on to add extra pipeline stage registers to the data path.
Bits per color sample	4-20, Default = **8**	Select the number of bits per color sample.
Number of inputs	**1** or 2	Select the number of inputs.

continued...

Parameter	Value	Description
Number of outputs	**1** or 2	Select the number of outputs.
din_*n*: Add input fifo	On or **Off**	Turn on if you want to add a FIFO at the input to smooth the throughput burstiness.
din_*n*: Input fifo size	1–128, Default = **8**	Specify the size (in powers of 2) of the input FIFO (in number of input beats).
din_*n*: Number of color planes	1–4, Default = **3**	Select the number of color planes per pixel.
din_*n*: Color planes transmitted in parallel	**On** or Off	Select whether the color planes are in parallel or in sequence (serially).
din_*n*: Number of pixels in parallel	1, 2, **4**	Specify the number of pixels received in parallel (per clock cycle).
din_*n*: Specify an input pattern over two pixels	On or **Off**	Turn on if you want to create an input color pattern using two consecutive input pixels instead of one.
din_*n*: Input pattern for pixel 0	—	Select a unique symbol name for each color plane of pixel 0. Each symbol may appear only once and must not be reused for pixel 1, or when specifying the color pattern for the other input.
din_*n*: Input pattern for pixel 1	—	Select a unique symbol name for each color plane of pixel 1. This parameter is only available if you turn on **Specify an input pattern over two pixels**.
dout_*n*: Add output fifo	On or **Off**	Turn on if you want to add a FIFO at the output to smooth the throughput burstiness.
dout_*n*: Output fifo size	1–128, Default = **8**	Specify the size (in powers of 2) of the output FIFO (in number of output beats).
dout_*n*: Number of color planes	1–4, Default = **3**	Select the number of color planes per pixel.
dout_*n*: Color planes transmitted in parallel	**On** or Off	Select whether to transmit the color planes in parallel or in sequence (serially).
dout_*n*: Number of pixels in parallel	1, 2, **4**	Specify the number of pixels transmitted in parallel (per clock cycle).
dout_*n*: Propagate user packets from input 0	—	Select whether user packets from input 0 are propagated through output *n*. This parameter is only available if you turn on **Pass all user packets through to the output(s)**.
dout_*n*: Propagate user packets from input 1	—	Select whether user packets from input 1 are propagated through output *n*. This parameter is only available if you turn on **Pass all user packets through to the output(s)** and **Specify an input pattern over two pixels**.
dout_*n*: Specify an output pattern over two pixels	On or **Off**	Turn on if you want to create an output color pattern using two consecutive output pixel instead of one.
dout_*n*: Output pattern for pixel 0	—	Select a valid symbol name for each color plane of pixel 0. The symbol must be defined on one of the input color patterns.
dout_*n*: Output pattern for pixel 1	—	Select a valid symbol name for each color plane of pixel 1. The symbol must be defined on one of the input color patterns. This parameter is only available if you turn on **Specify an output pattern over two pixels**.

12. Color Space Converter II IP Core

The Color Space Converter II IP core transforms video data between color spaces. The color spaces allow you to specify colors using three coordinate values.

You can configure this IP core to change conversion values at run time using an Avalon-MM slave interface.

The Color Space Converter II IP core offers the following features:

- Provides a flexible and efficient means to convert image data from one color space to another.
- Supports a number of predefined conversions between standard color spaces.
- Allows the entry of custom coefficients to translate between any two three-valued color spaces.
- Supports up to 4 pixels per transmission.

A color space is a method for precisely specifying the display of color using a three-dimensional coordinate system. Different color spaces are best for different devices, such as R'G'B' (red-green-blue) for computer monitors or Y'CbCr (luminance-chrominance) for digital television.

Color space conversion is often necessary when transferring data between devices that use different color space models. For example, to transfer a television image to a computer monitor, you are required to convert the image from the Y'CbCr color space to the R'G'B' color space. Conversely, transferring an image from a computer display to a television may require a transformation from the R'G'B' color space to Y'CbCr.

Different conversions may be required for standard definition television (SDTV) and high definition television (HDTV). You may also want to convert to or from the Y'IQ (luminance-color) color model for National Television System Committee (NTSC) systems or the Y'UV (luminance-bandwidth-chrominance) color model for Phase Alternation Line (PAL) systems.

12.1. Input and Output Data Types

The inputs and outputs of the Color Space Converter II IP core support signed or unsigned data and 4 to 20 bits per pixel per color plane. The IP cores also support minimum and maximum guard bands.

The guard bands specify ranges of values that must never be received by, or transmitted from the IP core. Using output guard bands allows the output to be constrained, such that it does not enter the guard bands.

12.2. Color Space Conversion

You convert between color spaces by providing an array of nine coefficients and three summands that relate the color spaces.

You can set these coefficients and summands at compile time, or you can enable the Avalon-MM slave interface to change them dynamically at run-time.

Given a set of nine coefficients [A0, A1, A2, B0, B1, B2, C0, C1, C2] and a set of three summands [S0, S1, S2], the IP cores calculate the output values for color planes 0, 1, and 2 (denoted dout_0, dout_1, and dout_2):

```
dout_0 = (A0 × din_0) + (B0 × din_1) + (C0 × din_2) + S0
dout_1 = (A1 × din_0) + (B1 × din_1) + (C1 × din_2) + S1
dout_2 = (A2 × din_0) + (B2 × din_1) + (C2 × din_2) + S2
```

Note: din_0, din_1, and din_2 are inputs read from color planes 0, 1, and 2.

The Color Space Converter II IP core supports the following predefined conversions that are available through the Platform Designer presets:

- Computer B'G'R' to CbCrY': SDTV
- CbCrY': SDTV to Computer B'G'R'
- Computer B'G'R' to CbCrY': HDTV
- CbCrY': HDTV to Computer B'G'R'
- Studio B'G'R' to CbCrY': SDTV
- CbCrY': SDTV to Studio B'G'R'
- Studio B'G'R' to CbCrY': HDTV
- CbCrY': HDTV to Studio B'G'R'
- IQY' to Computer B'G'R'
- Computer B'G'R' to IQY'
- UVY' to Computer B'G'R'
- Computer B'G'R' to UVY'

The values are assigned in the order indicated by the conversion name. For example, if you select Computer B'G'R' to CbCrY': SDTV, din_0 = B', din_1 = G', din_2 = R', dout_0 = Cb', dout_1 = Cr, and dout_2 = Y'.

If the channels are in sequence, din_0 is first, then din_1, and din_2. If the channels are in parallel, din_0 occupies the least significant bits of the word, din_1 the middle bits, and din_2 the most significant bits. For example, if there are 8 bits per sample and one of the predefined conversions inputs B'G'R', din_0 carries B' in bits 0–7, din_1 carries G' in bits 8–15, and din_2 carries R' in bits 16–23.

12.2.1. Predefined Conversions

Predefined conversions only support unsigned input and output data.

If you select signed input or output data, the predefined conversion produces incorrect results. When using a predefined conversion, the precision of the coefficients and summands must still be defined.

Predefined conversions are only defined for input and output bits per pixel per color plane equal to 8, 10, and 12. You must manually scale the summands accordingly when using a different bits per color plane value. If you use different input and output bits per pixel per color plane, you must also shift the results by the correct number of binary places to compensate. For example, to convert from 10-bit CbCrY' to 8-bit Computer B'G'R', select the conversion preset for 10-bit CbCrY' to 10-bit computer B'G'R'. The summands are already scaled for a 10-bit input so they remain unchanged. Change the output bits per color plane value from 10 to 8 on the parameter editor and follow the instructions of the warning message to shift the results by the correct number of binary places (2 places to the left).

Note: Always check the matrix of coefficients after applying a predefined conversion or after custom modifications. If the differences between the desired floating-point coefficient values and their actual fixed-point quantized values indicate an unacceptable loss of precision, you must increase the number of integer and/or fractional bits to fix the problem.

12.3. Result of Output Data Type Conversion

After the calculation, the fixed point type of the results must be converted to the integer data type of the output.

This conversion is performed in four stages, in the following order:

1. Result scaling—You can choose to scale up the results, increasing their range. This is useful to quickly increase the color depth of the output.
   - The available options are a shift of the binary point right –16 to +16 places.
   - This is implemented as a simple shift operation so it does not require multipliers.
2. Removal of fractional bits—If any fractional bits exist, you can choose to remove them:
   - Truncate to integer—Fractional bits are removed from the data. This is equivalent to rounding towards negative infinity.
   - Round-half up—Round up to the nearest integer. If the fractional bits equal 0.5, rounding is towards positive infinity.
   - Round-half even. Round to the nearest integer. If the fractional bits equal 0.5, rounding is towards the nearest even integer.
3. Conversion from signed to unsigned—If any negative numbers can exist in the results and the output type is unsigned, you can choose how they are converted:
   - Saturate to the minimum output value (constraining to range).
   - Replace negative numbers with their absolute positive value.
4. Constrain to range—logic that saturates the results to the minimum and maximum output values is automatically added:
   - If any of the results are not within the minimum and maximum values allowed by the output bits per pixel
   - If any of the results are beyond the range specified by the output Guard bands (optional)

12.4. Color Space Converter Parameter Settings

Table 37. Color Space Converter II Parameter Settings

Parameter	Value	Description
General		
Color planes transmitted in parallel	**On** or Off	Turn on to transmit the color planes in parallel.
Number of pixels in parallel	**1**, 2, 4, or 8	Specify the number of pixels transmitted or received in parallel.
Input data type: Input bits per pixel per color plane	4–20, Default = **8**	Specify the number of input bits per pixel (per color plane).
Input data type: Signed	On or **Off**	Turn to specify the output as signed 2's complement.
Input data type: Guard bands	On or **Off**	Turn to use a defined input range.
Input data type: Max	-524288–1048575, Default = **255**	Specify the input range maximum value.
Input data type: Min	-524288–1048575, Default = **0**	Specify the input range minimum value.
Output data type: Bits per pixel per color plane	4–20, Default = **8**	Select the number of output bits per pixel (per color plane).
Output data type: Signed	On or **Off**	Turn to specify the output as signed 2's complement.
Output data type: Guard bands	On or **Off**	Turn on to enable a defined output range.
Output data type: Max	-524288–1048575, Default = **255**	Specify the output range maximum value.
Output data type: Min	-524288–1048575, Default = **0**	Specify the output range minimum value.
How user packets are handled	• **No user packets allowed** • Discard all user packets received • Pass all user packets through to the output	If you design does not require the IP core to propagate user packets, then you may select to discard all user packets to reduce ALM usage. If your design guarantees there will never be any user packets in the input data stream, then you further reduce ALM usage by selecting **No user packets allowed**. In this case, the IP core may lock if it encounters a user packet.
Conversion method	**LSB** or MSB	This parameter is enabled when input and output bits per sample per color plane differ and when user packets are propagated. When the propagation of user packets requires padding or truncation, the IP can do one of the following: • Truncate or zero-pad the most significant bits • Truncate or pad the least significant bits
Run-time control	On or **Off**	Turn on to enable runtime control of the conversion values.
Reduced control register readback	On or **Off**	If you do not turn on this parameter, the values of all the registers in the control slave interface can be read back after they are written. If you turn on this parameter, the values written to registers 3 and upwards cannot be read back through the control slave interface. This option reduces ALM usage.

Operands		
Coefficient and summand fractional bits	0–31, Default = **8**	Specify the number of fraction bits for the fixed point type used to store the coefficients and summands.
Coefficient precision: Signed	**On** or Off	Turn on to set the fixed point type used to store the constant coefficients as having a sign bit.
Coefficient precision: Integer bits	0–16, Default = **1**	Specifies the number of integer bits for the fixed point type used to store the constant coefficients.
Summand precision: Signed	On or **Off**	Turn on to set the fixed point type used to store the constant summands as having a sign bit.
Summand precision: Integer bits	0–22, Default = **10**	Specifies the number of integer bits for the fixed point type used to store the constant summands.
Coefficients and Summand Table A0, B0, C0, S0 A1, B1, C1, S1 A2, B2, C2, S2	12 fixed-point values	Each coefficient or summand is represented by a white cell with a gray cell underneath. The value in the white cell is the desired value, and is editable. The value in the gray cell is the actual value, determined by the fixed-point type specified. The gray cells are not editable. You can create a custom coefficient and summand set by specifying one fixed-point value for each entry.
Move binary point right	-16 to +16, Default = **0**	Specify the number of places to move the binary point.
Remove fraction bits by	• **Round values - Half up** • Round values - Half even • Truncate values to integer	Select the method of discarding fraction bits resulting from the calculation.
Convert from signed to unsigned by	• **Saturating to minimum value at stage 4** • Replacing negative with absolute value	Select the method of signed to unsigned conversion for the results.

12.5. Color Space Conversion Control Registers

The width of each register in the Color Space Conversion control register map is 32 bits. To convert from fractional values, simply move the binary point right by the number of fractional bits specified in the user interface.

The control data is read once at the start of each frame and is buffered inside the IP core, so the registers can be safely updated during the processing of a frame.

Table 38. Color Space Converter II Control Register

The table below describes the control register map for Color Space Converter II IP core.

Address	Register	Description
0	`Control`	Bit 0 of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the IP core to stop the next time control information is read.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused.
2	`Interrupts`	Unused.
3	`Coeff-commit`	Writing a 1 to this location commits the writing of coefficient data. You must make this write to swap the coefficients currently in use with the latest set written to the register map.
4	`Coefficient A0`	The coefficient and summand registers use integer, signed 2's complement numbers. Refer to Color Space Conversion on page 109.

continued...

Address	Register	Description
5	Coefficient B0	
6	Coefficient C0	
7	Coefficient A1	
8	Coefficient B1	
9	Coefficient C1	
10	Coefficient A2	
11	Coefficient B2	
12	Coefficient C2	
13	Summand S0	
14	Summand S1	
15	Summand S2	

13. Chroma Resampler II IP Core

The Chroma Resampler II IP core resamples video data to and from common sampling formats.

The human eye is more sensitive to brightness than tone. Taking advantage of this characteristic, video transmitted in the Y'CbCr color space often subsamples the color components (Cb and Cr) to save on data bandwidth.

The Chroma Resampler II IP core allows you to change between 4:4:4 and 4:2:2 sampling rates where:

- 4:4:4 specifies full resolution in planes 1, 2, and 3 (Y, Cb and Cr respectively)
- 4:2:2 specifies full resolution in plane 1 and half width resolution in planes 2 and 3 (Y, Cb and Cr respectively)

Table 39. Chroma Resampler II Sampling Support for Algorithms

Sampling (4:2:2 to 4:4:4	Algorithm		
	Nearest Neighbor	Bilinear	Filtered
Upsampling	Yes	Yes	Yes
Downsampling	Yes	Yes	Yes
Horizontal Resampling	Yes	No	No
Vertical Resampling	Yes	No	No

You can configure the Chroma Resampler II IP core to operate in one of two generalized modes: fixed mode and variable mode.

Table 40. Chroma Resampler Modes

Fixed mode	Variable mode
• Both the input and output Avalon-ST Video interfaces are fixed to a set subsampling format, either 4:2:2 or 4:4:4. • Select either a fixed 4:4:4 to 4:2:2 downsampling operation, or a fixed 4:2:2 to 4:4:4 upsampling operation. • Enable fixed mode by setting the **Variable 3 color interface** parameter to **NEITHER**. • Does not support 4:2:0 format and does not provide the option for run-time control of the subsampling on either interface.	• Configure either the input or output Avalon-ST video interface as a variable subsampling interface. • Has 3 color planes per pixel and may transmit data formatted as 4:4:4, 4:2:2 or 4:2:0 data, with the selected subsampling option set at run time through the Avalon-MM Slave control interface. • Enable variable mode by setting the **Variable 3 color interface** parameter to **INPUT** or **OUTPUT**. *Note:* If you leave the **Variable 3 color interface** parameter by the default selection, the IP core retains a fixed subsampling format of either 4:4:4 or 4:2:2.

The variable mode is mainly to be use with interface standards, such as HDMI 2.0, which allow color space and subsampling to vary at run time. Because most of the VIP IP cores support only a fixed subsampling (either 4:4:4 or 4:2:2), the Chroma Resampler II IP core is allows a variable-to-fixed conversion (**Variable 3 color**

interface = **INPUT**) at the input to the processing pipeline, and a fixed-to-variable conversion (**Variable 3 color interface** = **OUTPUT**) at the output of the processing pipeline

13.1. Chroma Resampler Algorithms

The Chroma Resampler II IP core supports 3 different resampling algorithms.

These three algorithms vary in the level of visual quality provided, the chroma siting, and the resources required for implementation:

- Nearest Neighbor
- Bilinear
- Filtered

13.1.1. Nearest Neighbor

Nearest neighbor is the lowest quality resampling algorithm, with the lowest device resource usage.

- For horizontal downsampling (4:4:4 to 4:2:2), it simply drops every other Cb and Cr sample.
- For horizontal upsampling (4:2:2 to 4:4:4), it simply repeats each Cb and Cr sample.
- For vertical downsampling (4:2:2 to 4:2:0) the chroma data from every other video line is discarded.
- For vertical upsampling (4:2:0 to 4:2:2) the chroma data is repeated for two lines of luma data.

Figure 52. Nearest Neighbor for Horizontal Resampling

The nearest neighbor algorithm uses left siting (co-siting) for the 4:2:2 chroma samples—both the Cb and Cr samples from the even indexed Y samples are retained during downsampling.

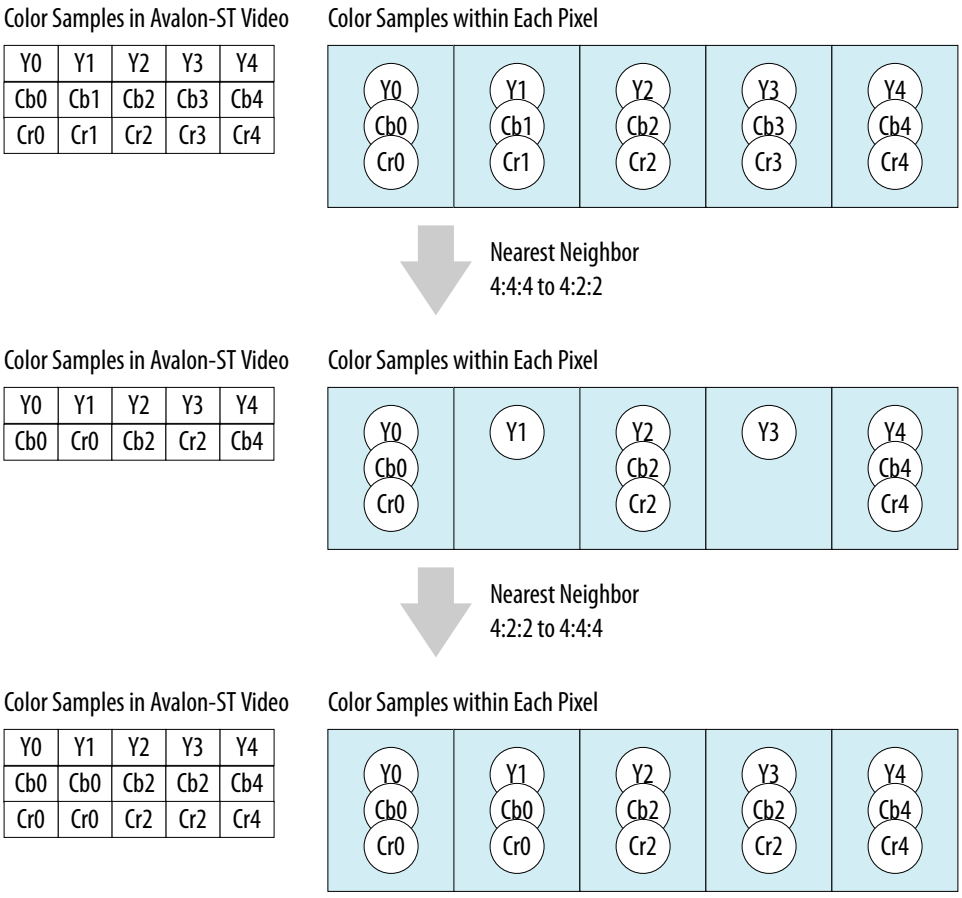
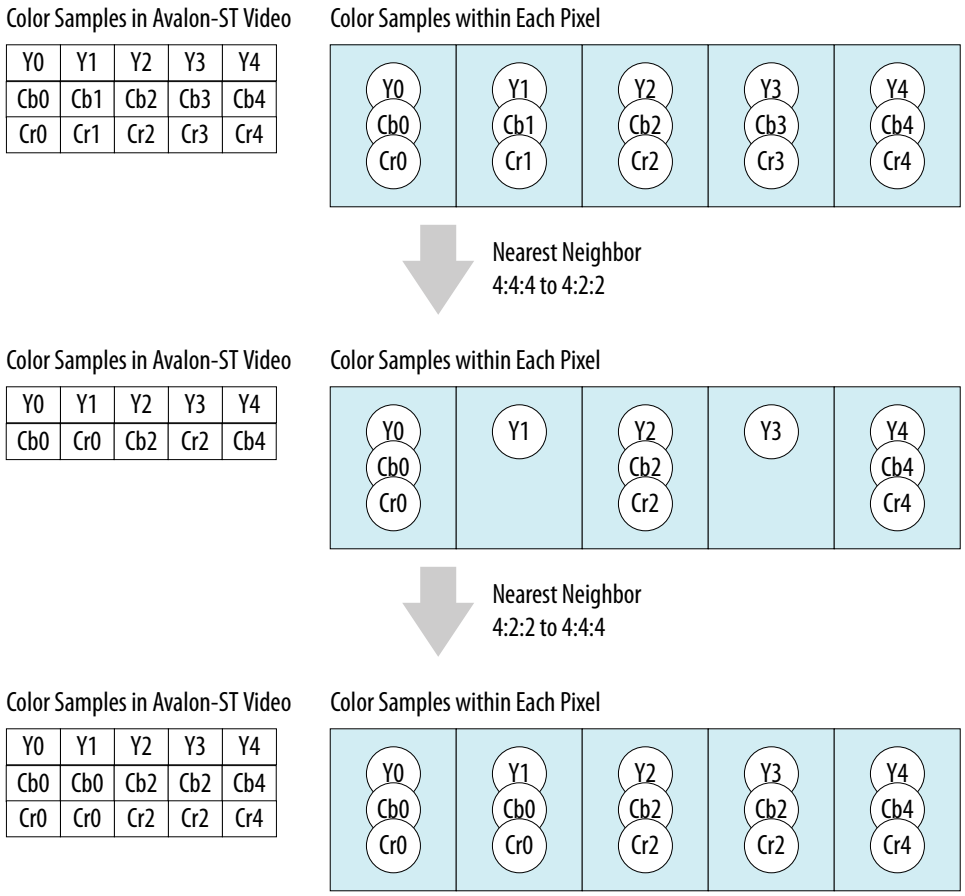


Figure 53. Nearest Neighbor for Vertical Resampling

The nearest neighbor algorithm uses top siting (co-siting) for both the Cb and Cr planes, for example, the chroma data from lines 0, 2, 4, and so on is preserved in downsampling, while the data from lines 1, 3, 5, and so on is discarded.



13.1.2. Bilinear

The bilinear algorithm offers a middle point between visual image quality and device resource cost.

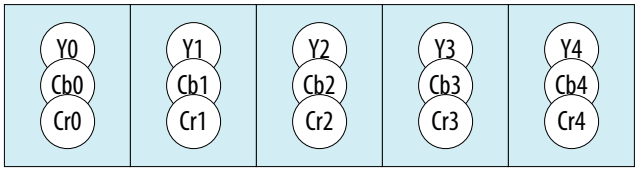
The figure and equations below show how the Chroma Resampler II IP core calculates the bilinear resampled chroma for both horizontal and vertical upsampling and downsampling. The bilinear algorithm uses center chroma siting for both the Cb and Cr samples in 4:2:2 format.

Figure 54. Bilinear Resampling

Color Samples in Avalon-ST Video

Y0	Y1	Y2	Y3	Y4
Cb0	Cb1	Cb2	Cb3	Cb4
Cr0	Cr1	Cr2	Cr3	Cr4

Color Samples within Each Pixel



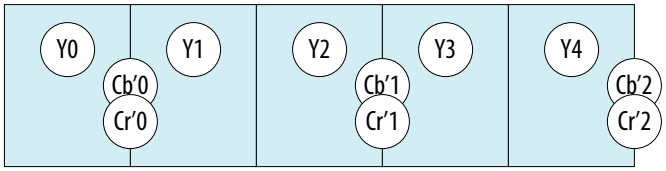
Bilinear
4:4:4 to 4:2:2

Color Samples in Avalon-ST Video

Y0	Y1	Y2	Y3	Y4
Cb'0	Cr'0	Cb'1	Cr'1	Cb'2

$Cb'i = (Cb(2 \times i) + Cb(2 \times I + 1)) / 2$
$Cr'i = (Cr(2 \times i) + Cr(2 \times I + 1)) / 2$

Color Samples within Each Pixel



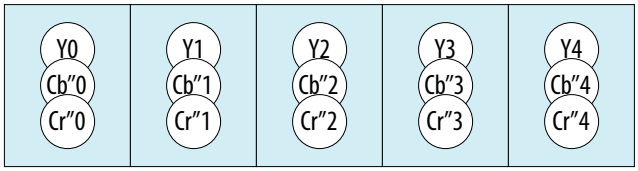
Bilinear
4:2:2 to 4:4:4

Color Samples in Avalon-ST Video

Y0	Y1	Y2	Y3	Y4
Cb"0	Cb"1	Cb"2	Cb"3	Cb"4
Cr"0	Cr"1	Cr"2	Cr"3	Cr"4

$i = 0, 2, 4, 5, ...$
$Cb"i = (3 \times Cb'(i/2) + Cb'(i/2 - 1)) /4$
$i = 1, 3, 5, 7, ...$
$Cr"i = (3 \times Cr'(i/2) + Cr'(i/2 - 1)) /4$

Color Samples within Each Pixel



13.1.3. Filtered

The filtered algorithm is the most computationally expensive and device resource heavy algorithm, but it offers increased visual quality.

You can parameterize the filtered algorithm to use either left siting (co-siting) or center siting of the chroma data.

- For downsampling conversions (4:4:4 to 4:2:2), the filtered algorithm applies an 8-tap Lanczos-2 resampling filter to generate the downsampled data. Different phase shifts are applied to the Lanczos-2 function when generating the coefficients, depending on the siting selected and whether the pixel index is even or odd. For left chroma siting, phase shifts of 0 and 0.5 are applied to the Lanczos-2 coefficients for the,even and odd indexed chroma samples respectively. For center chroma siting, the phases shifts are –0.25 and +0.25.
- For upsampling conversions (4:2:2 to 4:4:4), the filtered algorithm applies a 4-tap Lanczos-2 resampling filter to generate the upsampled data. For left chroma siting phase shifts of 0 and 0.5 are applied to the Lanczos-2 coefficients for the even and odd indexed chroma samples respectively. For center chroma siting the phases shifts are -0.25 and +0.25.

You may also opt to enable luma adaption for upsampling conversions. This feature further increases device resource usage (and is the only chroma resampler mode to implement some logic in DSP blocks), but may reduce color bleed around edges when compared to the default filtered algorithm.

When you enable luma adaption, the differences between successive luma samples are computed and compared to an edge threshold to detect significant edges. In areas where edges with strong vertical components are detected the phase of the Lanczos-2 filter can be shifted by up to 0.25 to the left or right to weight the resulting chroma samples more heavily towards the more appropriate side of the edge.

13.2. Chroma Resampler Parameter Settings

Table 41. Chroma Resampler II Parameter Settings

Parameter	Value	Description
Horizontal resampling algorithm	• NEAREST_NEIGHBOR • **BILINEAR** • FILTERED	Select the horizontal resampling algorithm to be used.
Horizontal chroma siting	• **LEFT** • CENTER	Select the horizontal chroma siting to be used. This option is only available for the filtered algorithm. The nearest neighbor algorithm forces left siting and bilinear algorithm forces center siting.
Enable horizontal luma adaptive resampling	On or **Off**	Turn on to enable horizontal luma-adaptive resampling. The parameter is only available for filtered upsampling.
Vertical resampling algorithm	• NEAREST_NEIGHBOR • **BILINEAR** • FILTERED	Select the vertical resampling algorithm to be used.
Vertical chroma siting	• **LEFT** • CENTER	Select the vertical chroma siting to be used. This option is only available for the filtered algorithm. The nearest neighbor algorithm forces top siting and bilinear algorithm forces center siting.

continued...

Parameter	Value	Description
Enable vertical luma adaptive resampling	On or **Off**	Turn on to enable vertical luma-adaptive resampling. The parameter is only available for filtered upsampling.
Maximum frame width	32–8192, Default = **1920**	Specify the maximum frame width allowed by the IP core.
Maximum frame height	32–8192, Default = **1080**	Specify the maximum frame height allowed by the IP core.
How user packets are handled	• No user packets allowed • Discard all user packets received • **Pass all user packets through to the output**	• If your design does not require the Chroma Resampler II IP core to propagate user packets, then you may select **Discard all user packets received** to reduce ALM usage. • If your design guarantees that the input data stream will never have any user packets, then you can further reduce ALM usage by selecting **No user packets allowed**. In this case, the IP core may lock if it encounters a user packet.
Add extra pipelining registers	On or **Off**	Turn on to add extra pipeline stage registers to the data path. You must to turn on this option to achieve: • Frequency of 150 MHz for Cyclone V devices • Frequencies above 250 MHz for Arria V, Stratix V, Intel Arria 10, or Intel Cyclone 10 GX devices
Bits per color sample	4–20, Default = **8**	Select the number of bits per color plane per pixel.
Number of color planes	1–4, Default = **2**	Select the number of color planes per pixel.
Color planes transmitted in parallel	**On** or Off	Select whether to send the color planes in parallel or in sequence (serially).
Input pixels in parallel	1, 2, 4, 8, Default = **1**	Select the number of pixels transmitted per clock cycle on the input interface.
Output pixels in parallel	1, 2, 4, 8, Default = **1**	Select the number of pixels transmitted per clock cycle on the output interface.
Variable 3 color interface	• **NEITHER** • INPUT • OUTPUT	Select which interface uses the variable subsampling 3 color interface.
Enable 4:4:4 input	On or **Off**	Turn on to select 4:4:4 format input data. *Note:* The input and output formats must be different. A warning is issued when the same values are selected for both.
Enable 4:2:2 input	**On** or Off	Turn on to select 4:2:2 format input data. *Note:* The input and output formats must be different. A warning is issued when the same values are selected for both. The IP core does not support odd heights or widths in 4:2:2 mode.
Enable 4:2:0 input	On or **Off**	Turn on to select 4:2:0 format input data. *Note:* The input and output formats must be different. A warning is issued when the same values are selected for both.
Enable 4:4:4 output	**On** or Off	Turn on to select 4:4:4 format output data. *Note:* The input and output formats must be different. A warning is issued when the same values are selected for both.
Enable 4:2:2 output	On or **Off**	Turn on to select 4:2:2 format output data.

continued...

Parameter	Value	Description
		Note: The input and output formats must be different. A warning is issued when the same values are selected for both. The IP core does not support odd heights or widths in 4:2:2 mode.
Enable 4:2:0 output	On or **Off**	Turn on to select 4:2:0 format output data. *Note:* The input and output formats must be different. A warning is issued when the same values are selected for both.

13.3. Chroma Resampler Control Registers

Table 42. Chroma Resampler II Control Register Map

The Chroma Resampler II IP Core automatically includes an Avalon-MM control slave interface if you select GUI parameters that have either a variable format input or variable format output interface. For variable format interfaces, you select the required input or output subsampling format through the control slave. If both interfaces are fixed formats, then there are no configurable features so the control slave is omitted.

Note: As is the convention with all VIP Suite cores, when a control slave interface is included, the core resets into a stopped state and must be started by writing a '1' to the Go bit of the control register before any input data is processed.

Address	Register	Description
0	`Control`	Bit 0 of this register is the Go bit, all other bits are unused. Setting this bit to 0 causes the IP core to stop at the end of the next frame/field packet.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused. The Chroma Resampler II IP core sets this address to 0 between frames. It is set to 1 while the IP core is processing data and cannot be stopped.
2	`Interrupt`	This bit is not used because the IP core does not generate any interrupts.
3	`Selected subsampling`	Control the selected subsampling format on either the input or output interface (whichever is variable). Write 0 to select 4:2:0, 1 for 4:2:2, and 2 for 4:4:4.

14. Control Synchronizer IP Core

The Control Synchronizer IP core synchronizes the configuration change of IP cores with an event in a video stream. For example, the IP core can synchronize the changing of a position of a video layer with the changing of the size of the layer.

The Control Synchronizer IP core has the following ports:

- Avalon Video Streaming Input and Output port—passes through Avalon-ST Video data, and monitors the data for trigger events.
- Avalon Master port—writes data to the Avalon Slave control ports of other IP cores when the Control Synchronizer IP core detects a trigger event.
- Avalon Slave port—sets the data to be written and the addresses that the data must be written to when the IP core detects a trigger event.
- Avalon Slave Control port—disables or enables the trigger condition. You can configure the IP core before compilation to disable this port after every trigger event; disabling this port is useful if you want the IP core to trigger only on a single event.

The following events trigger the Control Synchronizer IP core:

- the start of a video data packet.
- a change in the width or height field of a control data packet that describes the next video data packet.

When the Control Synchronizer IP core detects a trigger event, the following sequence of events take place:

1. The IP core immediately stalls the Avalon-ST video data flowing through the IP core.
2. The stall freezes the state of other IP cores on the same video processing data path that do not have buffering in between.
3. The IP core then writes the data stored in its Avalon Slave register map to the addresses that are also specified in the register map.
4. After writing is complete, the IP core resumes the Avalon-ST video data flowing through it. This ensures that any cores after the Control Synchronizer IP core have their control data updated before the start of the video data packet to which the control data applies.
5. When all the writes from a Control Synchronizer IP core trigger are complete, an interrupt is triggered or is initiated, which is the “completion of writes” interrupt.

14.1. Using the Control Synchronizer IP Core

The example illustrates how the Control Synchronizer IP Core is set to trigger on the changing of the width field of control data packets.

In the following example, the Control Synchronizer IP Core is placed in a system containing the following IP cores:

- Test Pattern Generator II
- Frame Buffer II
- Scaler II

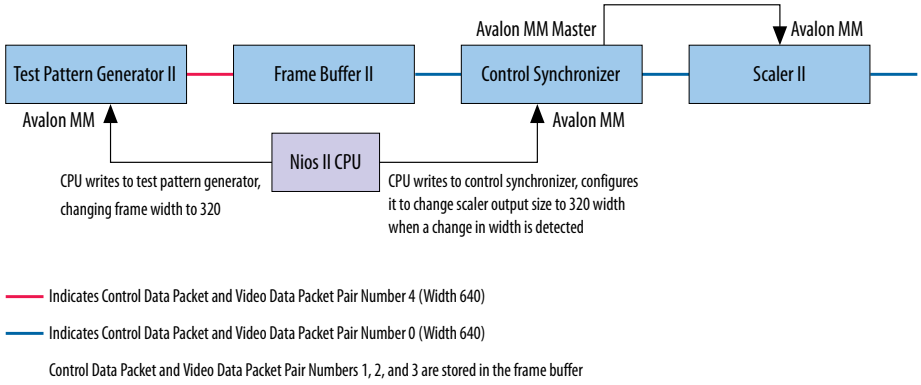
The Control Synchronizer IP core must synchronize a change of the width of the generated video packets with a change to the scaler output size in the following conditions:

- The scaler maintains a scaling ratio of 1:1 (no scaling)
- The frame buffer is configured to drop and repeat making it impossible to calculate packets streamed into the frame buffer and streamed out to the scaler.
- The scaler cannot be configured in advance of a certain video data packet.

The Control Synchronizer IP Core solves the problem through the following sequence of events:

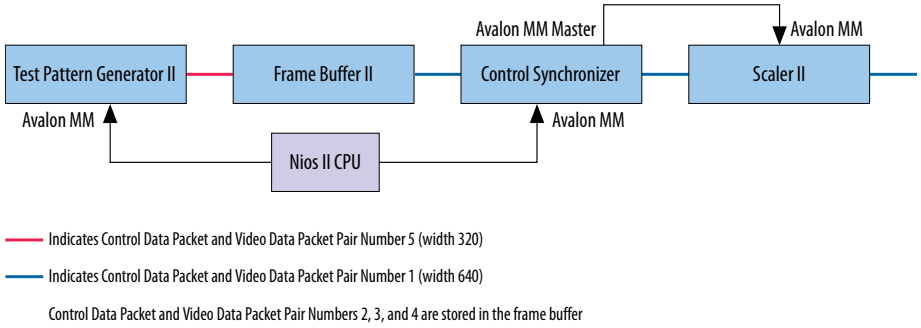
1. Sets up the change of video width.

Figure 55. Change of Video Width



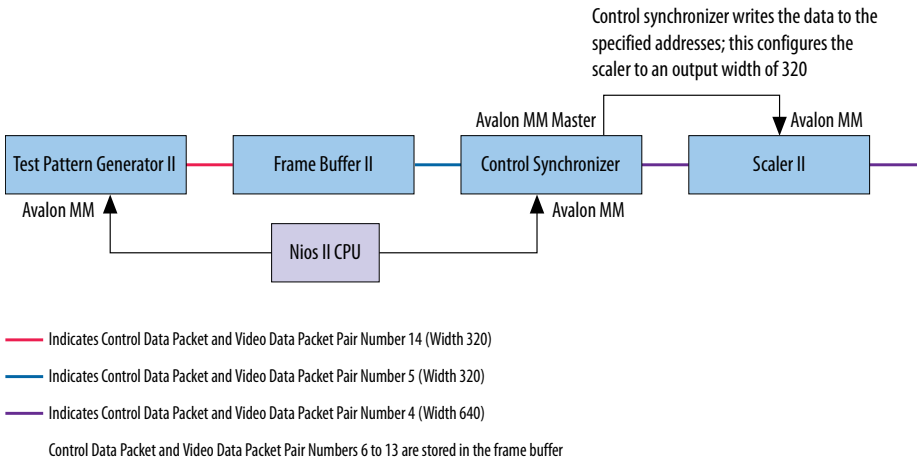
2. The test pattern generator changes the size of its Video Data Packet and Control Data Packet pairs to 320 width. It is not known when this change will propagate through the frame buffer to the scaler.

Figure 56. Changing Video Width



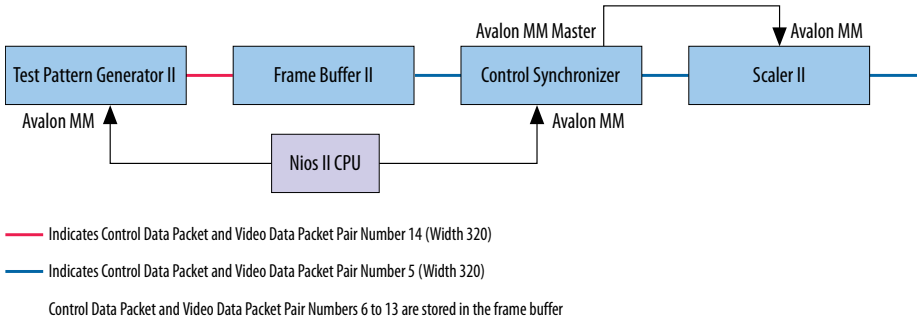
3. The Video Data Packet and Control Data Packet pair with changed width of 320 propagates through the frame buffer. The control synchronizer detects the change and triggers a write to the scaler. The control synchronizer stalls the video processing pipeline while it performs the write.

Figure 57. Test Pattern Generator Change



4. The scaler is reconfigured to output width 320 frames. The control synchronizer resumes the video processing pipeline. The scaling ratio maintains at 1:1.

Figure 58. Reconfigured Scaler II



14.2. Control Synchronizer Parameter Settings

Table 43. Control Synchronizer Parameter Settings

Parameter	Value	Description
Bits per pixel per color plane	4-20, Default = **8**	Select the number of bits per pixel (per color plane).
Number of color planes	1–4, Default = **3**	Select the number of color planes that are sent over one data connection. For example, a value of 3 for R'G'B' R'G'B' R'G'B' in serial.
Color planes are in parallel	**On** or Off	• Turn on to set colors planes in parallel. • Turn off to set colors planes in series.
Trigger on width change	**On** or Off	Turn on to start transfer of control data when there is a change in width value.
Trigger on height change	**On** or Off	Turn on to start transfer of control data when there is a change in height value.
Trigger on start of video data packet	On or **Off**	Turn on to start transfer of control data when the core receives the start of video data packet.
Require trigger reset via control port	On or **Off**	Turn on to disable the trigger once triggered. If you turn on this parameter, you need to enable the trigger using the control port.
Maximum number of control data entries	1–10, Default = **3**	Specify the maximum number of control data entries that can be written to other cores.

14.3. Control Synchronizer Control Registers

Table 44. Control Synchronizer Register Map

The control data is read once at the start of each frame and is buffered inside the IP core, so the registers can be safely updated during the processing of a frame.

Note: The width of each register of the frame reader is 32 bits.

Address	Register	Description
0	`Control`	• Bit 0 of this register is the Go bit. Setting this bit to 0 causes the IP core to start passing through data. • Bit 1 of this register is the interrupt enable. Setting this bit to 1 enables the completion of writes interrupt.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused.
2	`Interrupt`	Bit 1 of this register is the completion of writes interrupt bit, all other bits are unused. Writing a 1 to bit 1 resets the completion of writes interrupt.
3	`Disable Trigger`	• Setting this register to 1 disables the trigger condition of the control synchronizer. • Setting this register to 0 enables the trigger condition of the control synchronizer. When you turn on the **Require trigger reset via control port** parameter, this register value is automatically set to 1 every time the control synchronizer triggers.
4	`Number of writes`	This register sets how many write operations, starting with address and word 0, are written when the control synchronizer triggers.
5	`Address 0`	Address where word 0 must be written on trigger condition.
6	`Word 0`	The word to write to address 0 on trigger condition.

continued...

Address	Register	Description
7	Address 1	Address where word 1 must be written on trigger condition.
8	Word 1	The word to write to address 1 on trigger condition.
9	Address 2	Address where word 2 must be written on trigger condition.
10	Word 2	The word to write to address 2 on trigger condition.
11	Address 3	Address where word 3 must be written on trigger condition.
12	Word 3	The word to write to address 3 on trigger condition.
13	Address 4	Address where word 4 must be written on trigger condition.
14	Word 4	The word to write to address 4 on trigger condition.
15	Address 5	Address where word 5 must be written on trigger condition.
16	Word 5	The word to write to address 5 on trigger condition.
17	Address 6	Address where word 6 must be written on trigger condition.
18	Word 6	The word to write to address 6 on trigger condition.
19	Address 7	Address where word 7 must be written on trigger condition.
20	Word 7	The word to write to address 7 on trigger condition.
21	Address 8	Address where word 8 must be written on trigger condition.
22	Word 8	The word to write to address 8 on trigger condition.
23	Address 9	Address where word 9 must be written on trigger condition.
24	Word 9	The word to write to address 9 on trigger condition.

15. Deinterlacer II IP Core

The Deinterlacer II IP core (4K HDR passthrough) provides deinterlacing algorithms.

Interlaced video is commonly used in television standards such as phase alternation line (PAL) and national television system committee (NTSC), but progressive video is required by LCD displays and is often more useful for subsequent image processing functions.

The features for the Deinterlacer II IP core include:

- Run-time control and status registers to improve deinterlacing quality for mid-range motion-adaptive configurations
- Run-time control registers to allow run-time switching between bob, weave, and motion adaptive modes.
- Support for pass-through of progressive video at up to 4K resolutions and at a higher number of bits per pixel per color plane
- Integration of a stream cleaner core and embedded chroma resamplers where necessary

15.1. Deinterlacing Algorithm Options

The Deinterlacer II IP core is highly configurable. When using the IP core, choose the deinterlacing algorithm first, based on your design goals.

When you have selected the appropriate algorithm, it should be easy for you to determine the other parameters.

Table 45. Deinterlacing Algorithm Options

The table below provides some guidelines for you to consider when choosing the appropriate deinterlacing algorithm. All configurations support 1, 2, or 4 pixels in parallel.

Deinterlacing Algorithm	Quality	DDR Usage	Area	Latency	Film or Cadenced Content	Symbols in Sequence	Native 4K HDR/SDR Passthrough
Vertical Interpolation ("Bob")	Low	None	Low	1 line	Not supported	Supported	Supported
Field Weaving ("Weave")	Low	Low	Low	1 field	Not supported	Supported	Supported
Motion Adaptive	Medium	Medium	Low	1 line	3:2 and 2:2 detect and correct configurable	Not supported	Not supported
Motion Adaptive High Quality	High	High	High	1 field and 2 lines	3:2 with video over film and 2:2 detect and correct configurable	Not supported	Supported

DDR Usage:

- Low DDR usage—1 video field is read or written to DDR per output frame generated
- Medium DDR usage—approximately 4 fields of video is read or written to DDR per output frame generated
- High DDR usage—approximately 5 fields of video is read or written to DDR per output frame generated

Area:

- Low area—approximately 1–2K ALMs, ≤25 M10Ks, no DSP usage
- High area—approximately 15K ALMs, 44 DSPs

Quality:

- Low—some blockiness, flickering, or weave artifacts may be seen, depending on the content
- Medium—most content perceived as artifact-free, but some high frequency artifacts will be visible
- High—some tuning and software control may be required using the register set available, and then all content should display well, with minimal artifacts

Note: All deinterlacer configurations assume a new frame is starting if the height of the current field is different from the previous field. This means that if **NTSC deinterlacing** support is required, you must use a clipper to clip incoming fields of 244 lines of F0 and 243 lines of F1 input video, so no height difference is detected.

15.2. Deinterlacing Algorithms

The Deinterlacer II IP core provides four deinterlacing algorithms.

- Vertical Interpolation ("Bob")
- Field weaving ("Weave")
- Motion Adaptive
- Motion Adaptive High Quality (Sobel edge interpolation)

15.2.1. Vertical Interpolation (Bob)

The bob algorithm produces output frames by filling in the missing lines from the current field with the linear interpolation of the lines above and below them.

All color spaces and bits per pixel per color plane are supported.

At the top of an F1 field or the bottom of an F0 field there is only one line available so it is just duplicated. The function only uses the current field, therefore if the output frame rate is the same as the input frame rate, the function discards half of the input fields.

You can set the output frame rate (through the **Vertical Interpolation (Bob) deinterlacing behavior** parameter) to one of these options:

- **Produce one frame for every field**—interpolations are applied for each incoming field to create a new frame
- **Produce one frame for every F0 field** or **Produce one frame for every F1 field**—half the input field rate by producing fields on F0 or F1 according to the selection mode

15.2.2. Field Weaving (Weave)

Weave deinterlacing creates an output frame by filling all of the missing lines in the current field with lines from the previous field.

All color spaces and bits per pixel per color plane are supported.

This option gives good results for still parts of an image but unpleasant artifacts in moving parts. The weave algorithm requires external memory. This makes it significantly more expensive in external RAM bandwidth than the bob algorithms, if external buffering is not otherwise required. However, this option does not require any line buffering, making it the smallest deinterlacer configuration in terms of ALMs used.

Note: *Progressive segmented* video, where each video frame splits into two fields, may not perfectly deinterlace with the weave deinterlacer, because it is necessary to detect which field pairs belong together. To enable the detection of the pairs, select **2:2 detector** for the **Cadence detect and correction** parameter in motion adaptive configurations of the Deinterlacer II IP core.

15.2.3. Motion Adaptive

Motion Adaptive algorithm avoids the weaknesses of bob and weave algorithms by using bob deinterlacing for moving areas of the image and weave deinterlacing for still area.

All color spaces and bits per pixel per color plane are supported, although a YCbCr color space is used internally for high memory bandwidth configurations with video over film cadence detection.

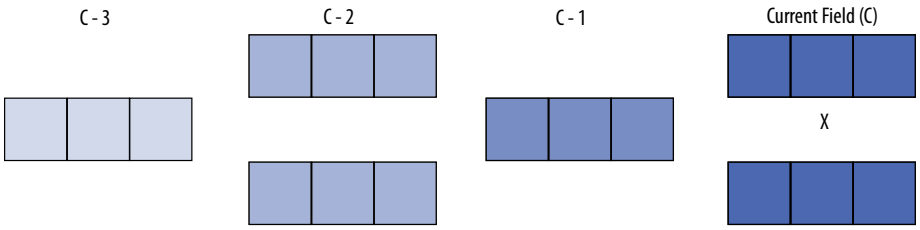
If the motion computed from the current and the previous pixels is higher than the stored motion value, the stored motion value is irrelevant. The function uses the computed motion in the blending algorithm, which then becomes the next stored motion value. However, if the computed motion value is lower than the stored motion value, the following actions occur:

- The blending algorithm uses the stored motion value.
- The next stored motion value is an average of the computed motion and of the stored motion.

This computed motion means that the motion that the blending algorithm uses climbs up immediately, but takes about four or five frames to stabilize. The motion-adaptive algorithm fills in the rows that are missing in the current field by calculating a function of other pixels in the current field and the three preceding fields as shown in the following sequence:

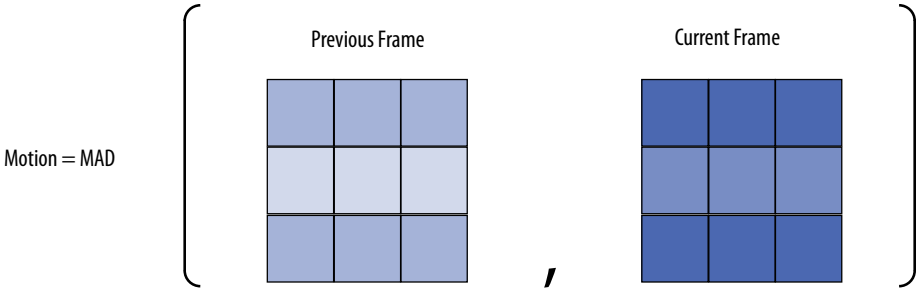
1. Pixels are collected from the current field and the three preceding it (the X denotes the location of the desired output pixel).

Figure 59. Pixel Collection for the Motion-Adaptive Algorithm



2. These pixels are assembled into two 3×3 groups of pixels. Figure 15–3shows the minimum absolute difference of the two groups.

Figure 60. Pixel Assembly for the Motion-Adaptive Algorithm



3. The minimum absolute difference value is normalized into the same range as the input pixel data. The function compares the motion value with a recorded motion value for the same location in the previous frame. If it is greater, the function keeps the new value; if the new value is less than the stored value, the function uses the motion value that is the mean of the two values. This action reduces unpleasant flickering artifacts.
4. The function uses a weighted mean of the interpolation pixels to calculate the output pixel and the equivalent to the output pixel in the previous field with the following equation:

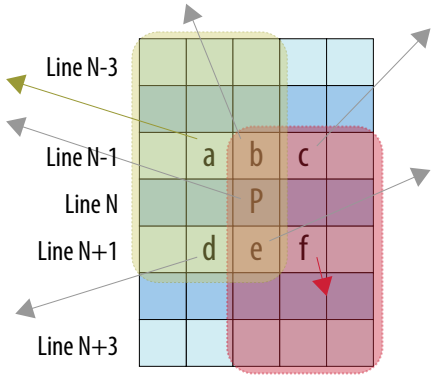
$$\text{Output Pixel} = M \times \frac{\text{Upper Pixel} + \text{Lower Pixel}}{2} + (1 - M) \times \text{Still Pixel}$$

15.2.4. Motion Adaptive High Quality (Sobel Edge Interpolation)

Motion Adaptive High Quality (Sobel edge interpolation) is the highest quality algorithm, applying a merged bob and weave based upon the amount of motion detected, and in areas of high motion applying a Sobel-based edge detection algorithm to interpolate between two pixels.

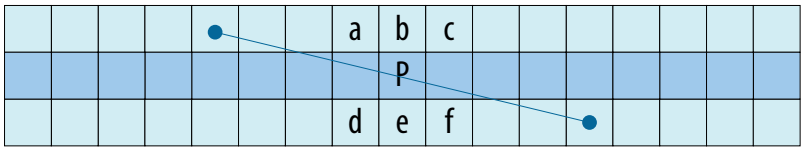
Figure 61. Sobel Edge Detection

The figure shows the kernel of pixels from which an interpolation decision is made.



- For the pixel being generated, P, in a missing line, N, from the current frame being generated, a kernel of 20 pixels is examined from lines N–3, N–1, N+1 and N+3.
- These 20 pixels are used to generate 7 smaller kernels over which Sobel transforms are performed (two of these are highlighted in yellow and red in the figure above).
- The Sobel transforms produce 7 motion vectors (as indicated by the arrows in the figure above), each comprised of a direction and magnitude.
- The deinterlacer uses this information to make the best possible interpolation over a wide kernel of 34 pixels taken from lines N-1 and lines N+1.

Figure 62. Sobel-based Edge Interpolation



15.3. Run-time Control

Enable run-time control if you require access to the register map.

If you do not select run-time control interface, the Deinterlacer II IP core starts deinterlacing as soon as it receives input video.

15.4. Pass-Through Mode for Progressive Frames

All configurations of the Deinterlacer II IP core support progressive passthrough of up to 1080p resolution. Some configurations support 4K progressive passthrough,

When progressive video passes through, there is no loss of precision or color space conversion internally.

15.5. Cadence Detection (Motion Adaptive Deinterlacing Only)

Motion-adaptive configurations of the Deinterlacer II IP core provide the option to detect both 3:2 and 2:2 cadences in the input video sequence, and perform a reverse telecine operation for perfect restoration of the original progressive video.

The **video over film** feature allows non-cadenced sections of video to be deinterlaced normally, regardless of the cadence. The **video over film** feature also enables enhanced scene change detection and a comprehensive register map for debugging and tuning the deinterlacer performance.

Note: Intel recommends you enable this feature for broadcast quality deinterlacing applications.

Figure 63. 2:2 Cadence (Progressive Segmented) Content

The figure below shows an example of four frames from a film; each frame is split into odd and even fields.

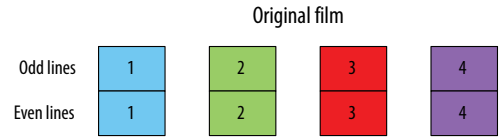
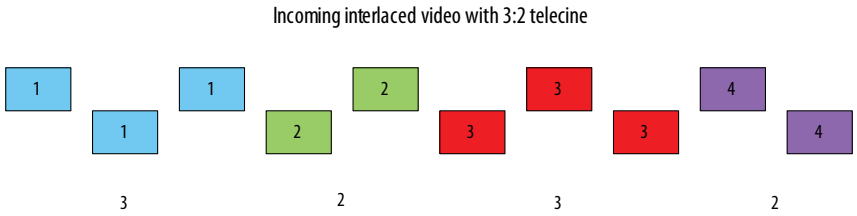


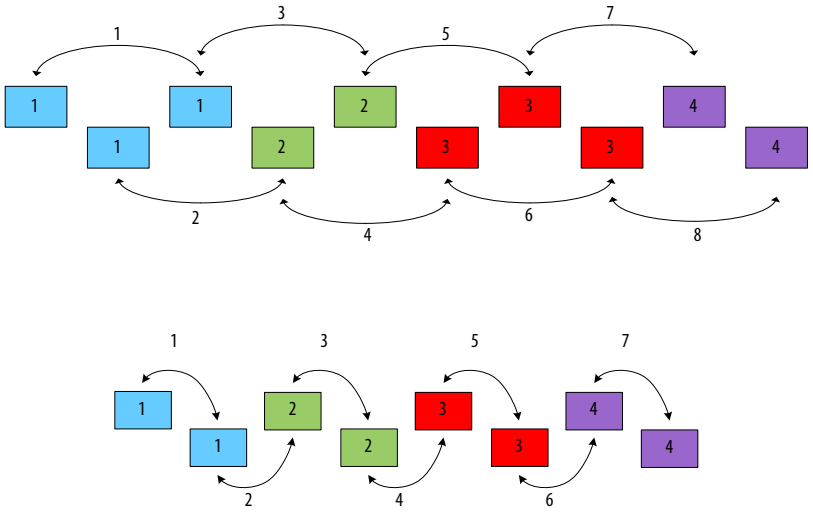
Figure 64. 3:2 Cadence



The Deinterlacer II handles such video sequence by detecting the cadence and reconstructing (reverse pulldown) the original film. This is achieved by comparing each field with the preceding field of the same type (3:2 detection) or detecting possible comb artifacts that occur when weaving two consecutive fields (2:2 detection).

Figure 65. 3:2 Detection and 2:2 Detection Comparison

The figure below shows the comparison between 3:2 and 2:2 detection.

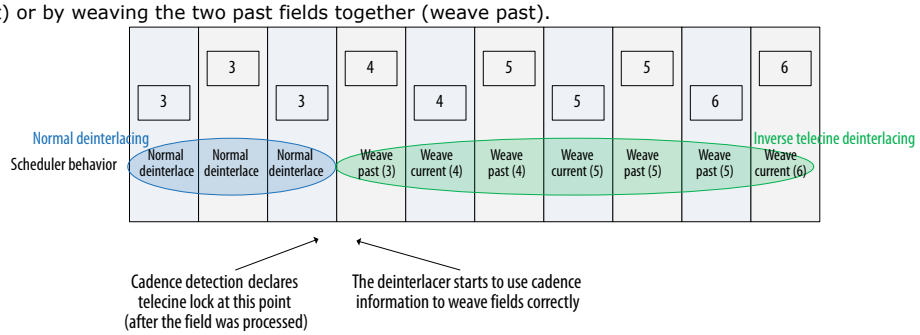


The 3:2 cadence detector tries to detect matches separated by four mismatches. When the 3:2 cadence detector sees this pattern a configurable number of times, it locks. The 3:2 cadence detector unlocks after configurable number of successive mismatches. Locking and unlocking thresholds for the 2:2 cadence detector are similarly configurable.

Refer to the *Deinterlacing Control Registers* section for more information.

Figure 66. Weave Current and Weave Past

When the cadence detect component enters a lock state, the deinterlacer continuously assembles a coherent frame from the incoming fields, by either weaving the current incoming field with the previous one (weave current) or by weaving the two past fields together (weave past).



If the incoming video contains any cadenced video, enable the **Cadence detection and reverse pulldown** option. Then, select the cadence detection algorithm according to the type of content you are expecting. If the incoming video contains both 3:2 and 2:2 cadences, select **3:2 & 2:2 detector**.

The cadence detection algorithms are also designed to be robust to false-lock scenarios—for example, features on adjacent fields may trick other detection schemes into detecting a cadence where there is none.

The Deinterlacer II IP core also provides **3:2 & 2:2 detector with video over film** option. Select this option to deinterlace correctly the subtitles, credits, or other closed-caption contents that were added over the top of movies or other cadenced contents.

Because this feature introduces a field of latency to allow weave operations to be performed either forwards or backwards, also set the **Fields buffered prior to output** to 1.

15.6. Avalon-MM Interface to Memory

Motion adaptive or weave deinterlacing algorithms require external memory storage, which may be configured as required.

The Deinterlacer II parameter editor calculates the top of the address space based on the configuration chosen.

15.7. Motion Adaptive Mode Bandwidth Requirements

The bandwidth usage for motion adaptive mode is 100% efficient.

For the example of 10bit 4:2:2 YCbCr video 1080i video, the requirements may be calculated as below.

- Image data:

  For every pair of output lines produced there are two phases:

  Phase 1: Read 2 lines = 1920 × 10 bits x 2 (YCbCr) × 2 = 76,800 bits per input line

  Phase 2: Write 1 line, Read 1 line, = 1920 × 10 bits × 2 × 2 = 76,800 bits per input line

  Phase 1 + phase 2 accesses = 153,600 bits of image data per input line

  153600 × 540 lines = 82,944,000 bits per output frame

  82944000 × 60 frames per second = 4,976,640,000 = 4.977 GBps of image data read/written

- Motion data:

  Motion data is always 8 bits per pixel regardless of color space.

  Read & Write motion = 1920 × 8 bits × 2 (one read and one write) = 30,720 bits per input line

  30,720 × 540 lines = 16,588,800 bits per output frame

  16,588,800 × 60 frames per second = 995,328,000 = 0.995 GBps of motion data written/read

- Video-over-film data:

  Motion data is always 24 bits per pixel regardless of color space.

  1920 × 24 bits × 2 (one read and one write per pixel) = 92,160 bits per input line

  92,160 × 540 lines = 49,766,400 bits per output frame

  49,766,400 × 60 frames per second = 2,985,984,000 = 2.985 GBps of video over film data written/read

Total bandwidth (without video over film cadence detection) = 4.977 + 0.995 = **5.972 GBps**

Total bandwidth (with video over film cadence detection) = 5.972 + 2.985 = **8.957 GBps**

Caution: Some memory configurations for motion adaptive configurations of the Deinterlacer II (excluding video over film configurations) can result in motion information being incorrectly overwritten by information from subsequent lines, resulting in a reduction in QOR manifesting in weave artifacts. If you observe weave artifacts in motion adaptive configurations, file an Intel Premier Support case on the Intel Support page.

15.8. Avalon-ST Video Support

You can configure the Deinterlacer II to accept interlaced content to a maximum width of 2048 and maximum height of 1080 (1080i).

Note: The Deinterlacer II IP core supports a maximum field width of 2048 pixels to allow the deinterlacing of progressive segmented frames of this width, as specified in SMPTE ST 2048-2:2011 Annex A.

Progressive content of all resolutions, including 4K content, will be passed through unchanged.

The Deinterlacer II always passes through user packets. The Deinterlacer II IP core contains an embedded Avalon-ST Stream Cleaner IP core for motion adaptive configurations, which you may disable. It is included to ensure that there is no possibility of malformed input packets locking up the deinterlacer.

You may disable the embedded stream cleaner but this is only recommended if a stream cleaner already exists in the system upstream of the deinterlacer or if you select one of the simpler algorithms (**Vertical interpolation ("Bob")** or **Field weaving ("Weave")**, which have complete resilience to all possible malformed inputs.

15.9. 4K Video Passthrough Support

The Deinterlacer II IP core offers different approaches for 4K video passthrough based on your deinterlacer configuration..

To determine the best approach for your deinterlacer configuration, refer to the table below.

Table 46. Appropriate Approaches for 4K (SDR and HDR) Passthrough

For Bob and Weave configurations, 4K HDR and SDR passthrough is natively supported. However, if you observe that the required f_{MAX} (e.g. 300 MHz for 2 pixels in parallel) cannot be reached, then use approach B to switch the deinterlacer to 4 pixels in parallel and half the f_{MAX}.

Deinterlacing Algorithm	Cadence Detection Algorithm	Approach
Bob	Not applicable	None — 4K passthrough supported natively
Weave	Not applicable	None — 4K passthrough supported natively

continued...

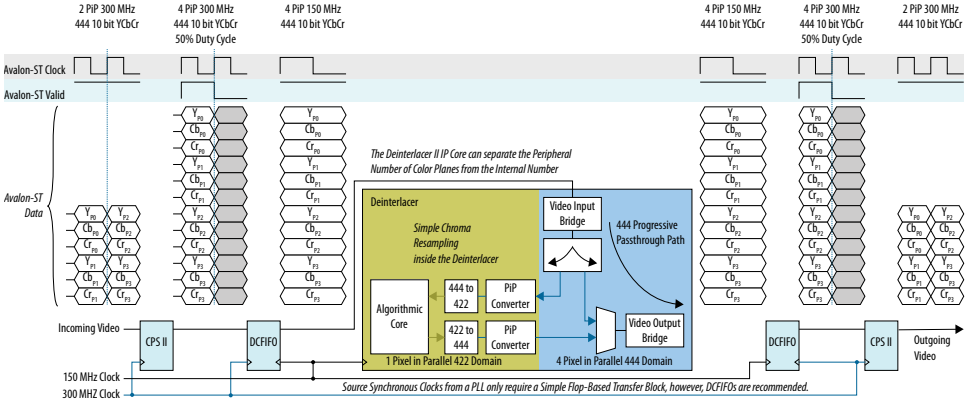
Deinterlacing Algorithm	Cadence Detection Algorithm	Approach
Motion adaptive	Any settings	Approach B — 4K passthrough not supported natively; use two Switch II IP instances
Motion adaptive high quality	3:2 and 2:2 detector with video over film	Approach A — 4K passthrough supported natively but conversion for f_{MAX} required
	Other settings	Approach B — 4K passthrough not supported natively; use two Switch II IP instances

15.9.1. Approach A

Approach A is for motion adaptive with 3:2 and 2:2 detector with video over film configurations where 4K HDR and SDR passthrough is natively supported but require f_{MAX} conversion.

Figure 67. 4K Video Passthrough (10-bit 4:4:4 YCbCr)

The figure below shows the recommended approach to convert the data coming in and out of the deinterlacer.



The following sequences describe the conversion:

1. The Color Plane Sequencer II IP core converts between 2 and 4 pixels in parallel.
2. Dual-clock FIFOs (Avalon-ST Dual Clock FIFO IP core) transfer data with a 50% duty cycle on the Avalon-ST valid signal in a 300MHz clock domain to 100% duty cycle in a 150MHz clock domain.
3. The deinterlacer accepts pixels in parallel data and converts any interlaced content to 1 pixel in parallel for deinterlacing.
4. Progressive content (and user packets) is maintained at the configured number of pixels in parallel and is unaffected by passing through the deinterlacer.

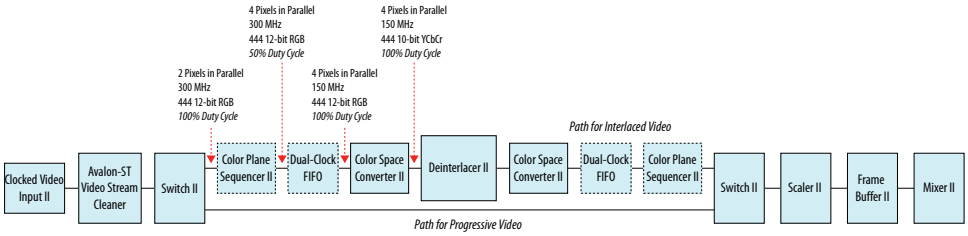
15.9.2. Approach B

Approach B is for motion adaptive configurations where 4K HDR and SDR passthrough is not natively supported.

For these configurations, you can use a pair of switches together with appropriate software control to ensure that any 4K content bypasses the deinterlacer.

Figure 68. Using Pair of Switches with Appropriate Software Control

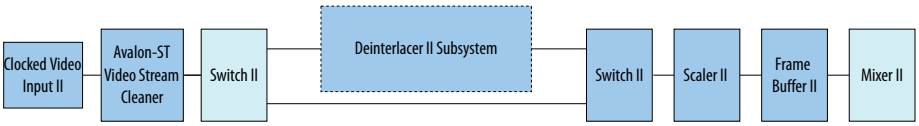
The Color Plane Sequencer II, Dual-clock FIFO, and Color Space Converter II instances are necessary to ensure the deinterlacer reaches the f_{MAX} target.



The software control should operate in the following sequence:

1. Upon start up, or on Clocked Video Input loss of lock, the software places the mixer and the first switch into consume mode.

Figure 69. Mixer and First Switch in Consume Mode



In this mode, the frame buffer continues to operate, absorbing any residual frames from the deinterlacer subsystem, and repeating the last valid output frame consumed by the mixer. The mixer produces its background layer. The first switch consumes any frames or fields, or parts of frames or fields from the Clocked Video Input II instance. Therefore, the deinterlacer does not observe any new frames or fields.

2. The software continues to monitor the status of the Clocked Video Input II instance. If the Clocked Video Input instance continues to detect and produce frames with consistent resolution, the software removes the IP cores from consume mode. The software then sets the two switches according to the resolution reported by the Clocked Video Input instance or for 4K progressive video.

Figure 70. Without Bypassing the Deinterlacer

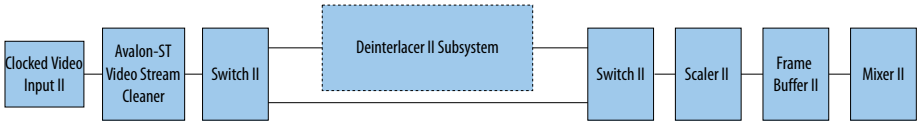
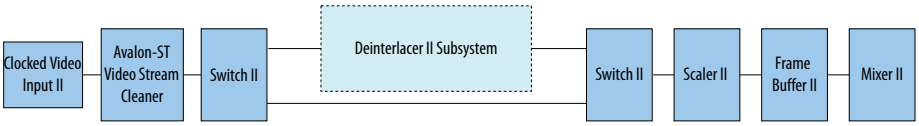


Figure 71. Bypassing the Deinterlacer for 4K Progressive Video



All Deinterlacer II variants support 1080p passthrough, so progressive resolutions of 1080p or less can either bypass the deinterlacer or not.

Approach B depends on the software control making the necessary switch CSR writes before any video packets reach the switch. The Clocked Video Input instance updates the resolution status registers at the end of the previous frame, so the software loop has a good portion of the vertical blanking interval to configure the video pipe. If little

or no blanking is present, the software could insert a 1-line FIFO at the start of the video pipe to allow for the necessary time required. Intel recommends that you place an Avalon-ST Video Stream Cleaner instance after the Clocked Video Input instance.

You can use a software control code similar to the example below for Approach B. In this example, a down-scaler follows the second switch; in lines 66-67 and 91-92 the scaler's status bit detects when all video has been flushed from the pipe. If a scaler is not present, the software may use the status bit for whichever component that follows the second switch for this purpose.

In this code, the following const declarations are assumed:

- VIP_HDMI_CVI_BASE - Address of the Clocked Video Input II
- VIP_CLEANER_BASE - Address of the Avalon-ST Video Stream Cleaner
- VIP_SWI_0_BASE - Address of the first (upstream) Switch II
- VIP_DIL_BASE - Address of the Deinterlacer II
- VIP_SWI_1_BASE - Address of the downstream Switch II
- VIP_SCALE_DOWN_BASE - Address of the down Scaler II
- VIP_VFB_BASE - Address of the triple-buffering Frame Buffer II
- VIP_MIXER_BASE - Address of the Mixer II

Example 1. Software Control Code

```
void start_cvi (unsigned int base) {
  unsigned int CVI_STAT=0;

  IOWR(base, 0, 1); // Starts the IP Core
  CVI_STAT=IORD(base, 1); // Read current status
  if ((CVI_STAT & 0x00000200)!=0) { // check the overflow
    IOWR(base, 1 , 0x00000200); // Reset the Overflow
  }
}

// read the current status of the video input
unsigned int current_cvi_status (unsigned int base) {
  unsigned int current_status;
  current_status = IORD(base, 1);
  return current_status;

  struct image_config {
   int x;
   int y;
   bool interlace;
   color_info image_color;
}

//-- Read the current image_config of the video input
image_config current_cvi_image_config (unsigned int cvi_base_addr) {

  image_config  current_image_config;
  unsigned int  current_status;
  unsigned int  interlaced_input;
  unsigned int  height_f1;
  int           color_pattern_reg;

  current_image_config.y = IORD(cvi_base_addr, 5); //-- Using f0 height only
  current_image_config.x = IORD(cvi_base_addr, 4);

  current_status   = current_cvi_status(cvi_base_addr);
  interlaced_input = ((current_status & 0x80)>>7);

  if (interlaced_input) {
```

```
    height_f1              = IORD(cvi_base_addr, 6);
    current_image_config.y = current_image_config.y + height_f1; // height is f0
+ f1
  }

  current_image_config.interlace = interlaced_input;

  color_pattern_reg = IORD(cvi_base_addr, 14);
  current_image_config.image_color.space =  color_pattern_reg & 0x00FF;
  current_image_config.image_color.depth = (color_pattern_reg & 0xFF00)>>8;

  if ( current_image_config.image_color.space==Y420_COLOR_SPACE ) {
     current_image_config.x = current_image_config.x << 1; // width is halved
for 4:2:0
  }

  return current_image_config;
}

void start_vip_core (unsigned int base) {
  IOWR(base,  0, 1);
}

void configure_dil_out_of_pipe() {
  unsigned int status;
  status = IORD(VIP_SWI_0_BASE, 1);
  status = IORD(VIP_SWI_1_BASE, 1);
  status = IORD(VIP_DIL_BASE, 1);

  IOWR(VIP_SWI_1_BASE,  0, 0); //Stop SWI0
  IOWR(VIP_SWI_0_BASE,  0, 0); //Stop SWI1

  //Wait until STATUS reflects STOPPED in the scaler:
  status = 1;
  while (status == 1) {
     status = IORD(VIP_SCALE_DOWN_BASE, 1);
  }

  IOWR(VIP_SWI_0_BASE,  4, 1); //SWI0 Output 0 control - ON
  IOWR(VIP_SWI_0_BASE,  5, 0); //SWI0 Output 1 control - OFF
  IOWR(VIP_SWI_1_BASE,  4, 1); //SWI1 Output 0 control - outputs from input 0
(passthru)
  IOWR(VIP_SWI_1_BASE,  0, 1); //Start SWI1
  IOWR(VIP_SWI_0_BASE,  0, 1); //Start SWI0
  IOWR(VIP_DIL_BASE,    0, 0); //Stop dil

}

void configure_dil_into_pipe() {
  unsigned int status;

  IOWR(VIP_SWI_1_BASE,  0, 0); //Stop SWI0
  IOWR(VIP_SWI_0_BASE,  0, 0); //Stop SWI1

  //Wait until STATUS reflects STOPPED:
  status = 1;
  while (status == 1) {
     status = IORD(VIP_SCALE_DOWN_BASE, 1);
  }

  IOWR(VIP_SWI_0_BASE,  4, 0); //SWI0 Output 0 control - OFF
  IOWR(VIP_SWI_0_BASE,  5, 1); //SWI0 Output 1 control - ON
  IOWR(VIP_SWI_1_BASE,  4, 2); //SWI1 Output 0 control - outputs from input 1
(dil)
  IOWR(VIP_SWI_1_BASE,  0, 1); //Start SWI1
  IOWR(VIP_DIL_BASE,    0, 1); //Start dil
  IOWR(VIP_SWI_0_BASE,  0, 1); //Start SWI0

}

int main() {
```

```
  const int NUM_VFB_FRAMES_TO_ABSORB = 10;  //Very safe,could be less
  int       vip_pipe_active          = 0;

  image_config prev_cvi_image_config;
  image_config cvi_image_config;
  image_config output_image_config;
  image_config requested_output_image_config;

  start_vip_core(VIP_HDMI_CVI_BASE);
  start_vip_core(VIP_CLEANER_BASE);
  start_vip_core(VIP_DIL_BASE);
  start_vip_core(VIP_VFB_BASE);
  start_vip_core(VIP_SCALE_DOWN_BASE);

  configure_dil_out_of_pipe();

  int vfb_frame_counter_prev = 0;
  int vfb_frame_counter      = 0;
  int vfb_frame_counter_orig = 0;
  int dil_configured         = 0;
  int cleaned_frames_passed  = 0;

  while(1) {

  // Always keep a frame count:
  vfb_frame_counter_prev = vfb_frame_counter;
  vfb_frame_counter = IORD(VIP_VFB_BASE, 3);

  prev_cvi_image_config = cvi_image_config;
  cvi_status       = current_cvi_status(VIP_HDMI_CVI_BASE);
  cvi_image_config = current_cvi_image_config(VIP_HDMI_CVI_BASE);

  if (   ((cvi_status & 0x00000800)==0)
   || (cvi_image_config.x != prev_cvi_image_config.x)
   || (cvi_image_config.y != prev_cvi_image_config.y)
   || (cvi_image_config.x >  UHD_DIM.x)
   || (cvi_image_config.y >  UHD_DIM.y)
   || (cvi_image_config.image_color.space !=
prev_cvi_image_config.image_color.space)
   ) {
    // The CVI is unlocked or the incoming resolution has changed or resolution
    // is invalid.  Put mixer's VIP pipe input in to consume so that output keeps
    // running and display is clean.
    if (vip_pipe_active) {
      IOWR(VIP_MIXER_BASE, 10,   2); // input 0 consumed

    // Need to put the first SWI into consume mode too, because we cannot allow
any
    // resolutions >1080p into the dil:
      IOWR(VIP_SWI_0_BASE, 16,   1);

    vfb_frame_counter_orig = IORD(VIP_VFB_BASE, 3);
      }

    vip_pipe_active       = 0;
    cleaned_frames_passed = 0;
    dil_configured        = 0;

    } else if (((cvi_status & 0x00000C00) >> 10) == 3) {// CVI locked with valid
resolution

    if (vip_pipe_active == 0) {

    // Before pipe is started, configure switches:
    if (!dil_configured) {

    cvi_image_config = current_cvi_image_config(VIP_HDMI_CVI_BASE);
    if (cvi_image_config.interlace) {
        configure_dil_into_pipe();
        } else {
```

```
        configure_dil_out_of_pipe();
        }

     // It is safe to take the switch out of consume mode now:
        IOWR(VIP_SWI_0_BASE, 16,   0);

        dil_configured = 1;
        }

    input_is_4k = (cvi_image_config.x==3840)?1:0;
    vip_pipe_active = 1;

      }

    if (cleaned_frames_passed == 0) {
    if (vfb_frame_counter != vfb_frame_counter_prev) {
    if ((vfb_frame_counter - vfb_frame_counter_orig) >=
NUM_VFB_FRAMES_TO_ABSORB) {
    // Mixer path re-enabled after NUM_VFB_FRAMES_TO_ABSORB frames, as cleaned
    // frames should now be flushed through
          IOWR(VIP_MIXER_BASE, 10,   1);
          cleaned_frames_passed = 1;
          }
        }
      }

    }

    prev_cvi_image_config = cvi_image_config;

  } //-- while(1)

  return 0;
}
```

15.10. Behavior When Unexpected Fields are Received

The behavior of the deinterlacer assumes an uninterrupted sequence of pairs of interlaced fields (F0, F1, F0, …) each having the same height.

You can guarantee this behavior through the use of the integrated Avalon-ST Video Stream Cleaner or an external stream cleaner. Some video streams may not follow this rule, and the behavior of the deinterlacer is undefined in such cases.

Note: When the bob algorithm is used, and synchronization is done on a specific field (input frame rate = output frame rate), the IP core always discards the field that is constantly unused. The IP core uses the other field to build a progressive frame.

15.11. Handling of Avalon-ST Video Control Packets

In all parameterizations, the Deinterlacer II IP core generates a new and updated control packet just before the processed image data packet.

This packet contains the correct frame height and the proper interlace flag so that the following image data packet is interpreted correctly by the following IP cores.

Note: The Deinterlacer II IP core uses 0010 and 0011 to encode interlacing values into the generated Avalon-ST Video packets. These flags mark the output as being progressive and record information about the deinterlacing process, which can be used by other IP cores. For example, the Interlacer IP core uses this information to precisely extract the original interlaced fields from the deinterlaced frame. The interlacing is encoded as 0000 when the Deinterlacer II IP core passes a progressive frame through.

15.12. Deinterlacer II Parameter Settings

Table 47. Deinterlacer II Parameter Settings

Parameter	Value	Description
Maximum width of interlaced content	32–2048, Default = **1920**	Specify the maximum frame width of any interlaced fields. The maximum frame width is the default width at start-up.
Maximum height of the generated progressive output	32–1080, Default = **1080**	Specify the maximum progressive frame height in pixels. The maximum frame height is the default progressive height at start-up.
Disable embedded Avalon-ST Video stream cleaner	**On** or Off	Turn on this option only if your system can guarantee to always supply well-formed control and video packets of the correct length.
Number of pixels in parallel	1, 2, 4, or 8	Select the number of pixels to be transmitted every clock cycle.
Bits per color sample	4–20	Select the number of bits per pixel (per color plane).
Number of color planes	2 or 3	Select the number of color planes per pixel.
Color planes transmitted in parallel	**On** or Off	Select if the Avalon-ST symbols are being transmitted in parallel.
YCbCr	**On** or Off	Turn on if you are using YCbCr 4:2:2 data format.
4:2:2	On or **Off**	Turn on if you are using 4:2:2 data format. *Note:* 4:2:2 mode does not support odd frame widths and heights.
Deinterlacing algorithm	• Vertical interpolation ("Bob") • Field weaving ("Weave") • Motion Adaptive • **Motion Adaptive High Quality (Sobel edge interpolation)**	Select the deinterlacing algorithm you want to use.
Vertical interpolation ("Bob") deinterlacing behavior	• Produce one frame every F0 field • Produce one frame every F1 field • **Produce one frame every field**	Determines the rate at which frames are produced and which incoming fields are used to produce them. *Note:* Only relevant if you set the deinterlacing algorithm to **Vertical interpolation ("Bob")**.
Run-time control	**On** or Off	Turn on to enable run-time control of the deinterlacer. When you turn on this parameter, the Go bit gets deasserted by default. When you turn off this parameter, the Go is asserted by default. *Note:* Intel strongly recommends run-time control when in motion adaptive modes with **3:2 & 2:2 detector with video over film**.
Cadence detection algorithm and reverse pulldown	• 3:2 detector • 2:2 detector • 3:2 & 2:2 detector • **3:2 & 2:2 detector with video over film**	Select the cadence detection algorithm you want to use.
Fields buffered prior to output	0 or 1, Default = **1**	Either 0 or 1 field is buffered prior to output. You must select 1 field of buffering for video over film cadence detection modes. Other modes incur no fields of latency delay.

continued...

Parameter	Value	Description
Cadence detection and reverse pulldown	**On** or Off	Turn on to enable automatic cadence detection and reverse pulldown. *Note:* Cadenced content originates from movies or TV shows. Enable **Cadence detection and reverse pulldown** only if this content type is processed, otherwise disable this feature to save resources.
Avalon-MM master(s) local ports width	• 16 • 32 • 64 • 128 • **256** • 512	Specify the width of the Avalon-MM ports used to access external memory. It is recommended to match this width to the Avalon-MM width of your EMIF controller.
Use separate clock for the Avalon-MM master interface(s)	**On** or Off	Turn on to add a separate clock signal for the Avalon-MM master interface(s) so that they can run at a different speed to the Avalon-ST processing. The separation decouples the memory speed from the speed of the data path. Intel expects most applications to use separate Avalon-MM and Avalon-ST clock rates, so make sure this parameter is turned on.
Base address of storage space in memory	0–0×7FFFFFFF, Default = **0×00000000**	Select a hexadecimal address of the frame buffers in external memory.
Top of address space	**0×00ca8000**	For your information only. Top of the deinterlacer address space. Memory above this address is available for other components.
FIFO depth Write Master	8–512, Default = **64**	Select the FIFO depth of the Avalon-MM write master interface.
Av-MM burst target Write Master	2–256, Default = **32**	Select the burst target for the Avalon-MM write master interface.
FIFO depth EDI Read Master	8–512, Default = **64**	Select the FIFO depth of the edge-dependent interpolation (EDI) Avalon-MM read master interface.
Av-MM burst target EDI Read Master	2–256, Default = **32**	Select the burst target for EDI Avalon-MM read master interface.
FIFO depth MA Read Master	8–512, Default = **64**	Select the FIFO depth of the motion-adaptive (MA) Avalon-MM read master interface.
Av-MM burst target MA Read Master	2–256, Default = **32**	Select the burst target for MA Avalon-MM read master interface.
FIFO depth Motion Write Master	8–512, Default = **64**	Select the FIFO depth of the motion Avalon-MM write master interface.
Av-MM burst target Motion Write Master	2–256, Default = **32**	Select the burst target for the motion Avalon-MM write master interface.
FIFO depth Motion Read Master	8–512, Default = **64**	Select the FIFO depth of the motion Avalon-MM read master interface.
Av-MM burst target Motion Read Master	2–256, Default = **32**	Select the burst target for motion Avalon-MM read master interface.

15.13. Deinterlacing Control Registers

Deinterlacer II Control Register Maps

The tables below describe the Deinterlacer II IP core control register map for run-time control. The Deinterlacer II reads the control data once at the start of each frame and buffers the data inside the IP core. The registers may safely update during the processing of a frame. Use these registers in software to obtain the best deinterlacing quality.

Table 48. Deinterlacer II Control Register Map for Vertical Interpolation (Bob) and Field Weaving (Weave) Deinterlacing Algorithms

Address	Register	RO/RW	Description
0	`Control`	RW	Bit 0 of this register is the Go bit, all other bits are unused. Setting this bit to 0 causes the Deinterlacer II IP core to stop the next time that control information is read. When you enable run-time control, the Go bit gets deasserted by default. If you do not enable run-time control, the Go is asserted by default. Power on value: 0
1	`Status`	RO	Bit 0 of this register is the `Status` bit, all other bits are unused. • The Deinterlacer II IP core sets this address to 0 between frames when the Go bit is set to 0. • The Deinterlacer II IP core sets this address to 1 while the core is processing data and cannot be stopped. Power on value: 0
2	Reserved	–	This register is reserved for future use.

For motion-adaptive configurations, Intel recommends that you initially retain all values at their reset value, with the exception of the following:

- The `Motion Shift` and `Motion Scale` registers may require adjustments to correct for weave artifacts caused by insufficient motion sensitivity. Refer to Tuning Motion Shift and Motion Scale Registers on page 152 for more information.
- The cadence-related registers in Set A (primarily registers 8 and 9) may require adjustments to correct undetected cadences.

The tables below detail the run-time control registers for the motion adaptive configurations of the Deinterlacer II IP core. Configurations with video-over-film mode enabled use Set B registers, all other configurations use Set A registers.

Note: For Set A registers, if you have not configured the **Cadence detection and reverse pulldown** parameter, the cadence-related registers have no effect.

Table 49. Deinterlacer II Control Register Map for Motion-Adaptive Parameterizations Set A

Address	Register	RO/RW/WO	Width	Description
0	Go	RW	32	Setting this bit to 0 causes the Deinterlacer II IP core to stop the next time that control information is read. When you enable run-time control, the Go bit gets deasserted by default. If you do not enable run-time control, the Go is asserted by default.

continued...

Address	Register	RO/RW/WO	Width	Description
				Power on value: 0
1	`Status`	RO	32	• The Deinterlacer II IP core sets this address to 0 between frames when the Go bit is set to 0. • The Deinterlacer II IP core sets this address to 1 while the core is processing data and cannot be stopped. Power on value: 0
2	Reserved	–	–	This register is reserved for future use.
3	Unused	–	–	Unused

4	`3:2 Cadence Diff Count`	WO	8	If the IP core could not detect 3:2 cadences, poll this register to facilitate the tuning of 3:2 detection.
				For example:
				`while(1){ pollCount++; if (pollCount%350 == 0) { diffCount = IORD(VIP_DIL_BASE, 4); printf ("Dil diffCount = %0d\n", diffCount);}`
				The register returns the number of lines in the current field for which a difference is detected compared to the preceding field. A successfully detected 3:2 cadence generates output such as the following:
				`Dil diffCount = 253`
				`Dil diffCount = 253`
				`Dil diffCount = 0`
				`Dil diffCount = 248`
				`Dil diffCount = 249`
				`Dil diffCount = 244`
				`Dil diffCount = 245`
				`Dil diffCount = 0`
				`Dil diffCount = 240`
				`Dil diffCount = 241`
				`Dil diffCount = 237`
				`Dil diffCount = 239`
				`Dil diffCount = 0`
				`Dil diffCount = 236`
				`Dil diffCount = 235`
				`Dil diffCount = 233`
				You are required to experiment with the polling rate to poll at a rate of approximately once per field.

5	`3:2 Cadence Lock Threshold`	WO	8	The higher the threshold value, the more stringent the requirements for the deinterlacer to start performing reverse telecine deinterlacing. You may set lower threshold values for greater sensitivity to cadenced sequences. The default value is 6.
6	`3:2 Cadence Unlock Threshold`	WO	8	Set this register lower than the `3:2 Cadence Lock Threshold` register. The closer the value is to the `3:2 Cadence Lock Threshold` register, the less stringent the requirements for the deinterlacer to lose lock and stop performing reverse telecine deinterlacing. The default value is 4.
7	`3:2 Cadence Diff Threshold`	WO	8	The `3:2 Cadence Diff Count` register increments when the "score" from the cadence detector for a line is equal or more than the `3:2 Cadence Diff Threshold` register. The default value is 16.
8	`3:2 Cadence Diff Line Ratio`	WO	8	This register affects the proportion of lines within a field that are required to exhibit a cadence before a 3:2 cadence lock can be achieved. Shift the number of lines using this value: `Lines >> 3:2 Cadence Diff Threshold` The default value is 4.

continued...

Address	Register	RO/RW/WO	Width	Description
9	3:2 Cadence Diff Noise Suppresion	WO	8	Use this register to shift the result of difference calculation logic which is used to determine whether a line is being repeated: Diff result >> 3:2 Cadence Diff Noice Suppresion The default value is 5 to allow resilience against noise in the LSBs masking a cadence relationship.
10	2:2 Cadence Lock Threshold	WO	4	The higher the threshold value, the more stringent the requirements for the deinterlacer to start performing reverse telecine deinterlacing.
11	2:2 Cadence Unlock Threshold	WO	4	Set this register lower than the 2:2 Cadence Lock Threshold register. The closer the value is to the 2:2 Cadence Lock Threshold register, the less stringent the requirements for the deinterlacer to lose lock and stop performing reverse telecine deinterlacing. The default value is 4.
12	2:2 Cadence Comb Threshold	WO	8	A "comb count" for a field increments when the "score" from the 2:2 cadence detector for a line is ≥ 2:2 Cadence Comb Threshold. Its reset value is 16. Decrease this value to increase the deinterlacer's sensitivity to 22 cadences.
13	Unused	–	–	Unused
14	Motion Shift	WO	8	Specifies the amount of raw motion (SAD) data that is right-shifted. Shifting is used to reduce sensitivity to noise when calculating motion (SAD) data for both "bob" and "weave" decisions and cadence detection. *Note:* It is very important to set this register correctly for good deinterlacing performance. Tune this register in conjunction with the motion visualization feature. Higher values decrease sensitivity to noise when calculating motion, but may start to introduce weave artifacts if the value used is too high.
15	Unused	–	–	Unused
16	Visualize Motion Values	WO	3	• Set bit 0 (Visualize Motion Values) to color pixels where motion is detected with pink for debugging purposes. The greater the luminance of the color pink, the more motion is detected. • Set bit 1 (Bob-Only Mode) to temporarily force the deinterlacer to do a pure "bob" deinterlace regardless of motion. • Set bit 2 (Weave-Only Mode) to temporarily force the deinterlacer to do a pure "weave" deinterlace regardless of motion.

Table 50. Deinterlacer II Control Register Map for Motion-Adaptive with Video Over Film Parameterizations Set B

Address	Register	RO/RW/WO	Width	Description
0	Control	RW	32	Bit 0 of this register is the Go bit, all other bits are unused. Setting this bit to 0 causes the Deinterlacer II IP core to stop the next time that control information is read. When you enable run-time control, the Go bit gets deasserted by default. If you do not enable run-time control, the Go is asserted by default.

continued...

Address	Register	RO/RW/WO	Width	Description
				Power on value: 0
1	`Status`	RO	32	Bit 0 of this register is the `Status` bit, all other bits are unused. • The Deinterlacer II IP core sets this address to 0 between frames when the `Go` bit is set to 0. • The Deinterlacer II IP core sets this address to 1 while the core is processing data and cannot be stopped. Power on value: 0
2	Reserved	–	–	This register is reserved for future use.
3	`Cadence Detected`	RO	1	• When polled, the least significant bit (LSB) to 1, indicates the Deinterlacer II IP core has detected a 3:3 or 2:2 cadence and is performing reverse telecine. • Bit 0 indicates otherwise. Range: 0–1 Power on value: 0
4	`Extra Status`	RO	32	• Bits [1:0]: 0 = Interlaced, 1 = Weave_current, 2 = Weave_future, 3 = Unused • Bit 2: VOF Lock Delay Up. The IP core sets this bit when the VOF lock delay period is over. • Bit 3: Adequate VOF region is detected. • Bit 4: Cadence 22 mode. The IP cores sets this bit when in 2:2 mode.
5	`3:2 Cadence Film Pixels locked`	RO	32	Number of pixels displaying film content in a given field. Range: 0–(2^{32}–1) Power on value: 0
6	`Motion in field`	RO	32	Total motion detected in the current field, computed from the sum of absolute differences (SAD) in Luma to the previous field of the same type, plus the Luma SAD of the previous field, and the next field, divided by 16. Range: 0–(2^{32}–1) Power on value: 0
7	`3:2 Cadence VOF Histogram Total Phase 1`	RO	32	Histogram of locked pixels, that is used for debugging purposes before the VOF lock. Indicates the number of pixels showing the presence of a potential cadence for this phase. If one phasing shows more pixels with a cadence present compared to other phasing by a factor 4 or more, all pixels in the field will be locked. Reverse telecine on per-pixel basis will commence VOF `Lock Delay` fields after the lock. Range: 0–(2^{32}–1) Power on value: 0
8	`3:2 Cadence VOF Histogram Total Phase 2`		32	
9	`3:2 Cadence VOF Histogram Total Phase 3`		32	
10	`3:2 Cadence VOF Histogram Total Phase 4`		32	
11	`3:2 Cadence VOF Histogram Total Phase 5`		32	

continued...

Address	Register	RO/RW/WO	Width	Description
12	Cadence Detect and advanced tuning registers On	RW	1	This register enables the cadence detection feature and (if configured) the video over film feature together with all the motion and cadence/VOF tuning registers. • Setting the LSB of this register to 1 enables cadence detection and tuning registers. • Setting the LSB of this register to 0 disables cadence detection and tuning registers. • Cadence detection is disabled on reset. Range: 0–1 Power on value: 0
13	Video Threshold	RW	8	The most important register to tune the video over film features. Set lower values for more emphasis on video and higher values for more emphasis on film. Set dynamically in software when the input content changes for best results. Range: 0–255 Power on value: 255
14	Film Lock Threshold	RW	24	• Bits 2:0 - Lock threshold for 3:2 cadence detection • Bits 10:8 - Lock threshold for 2:2 cadence detection • Bits 23:16 - Comb threshold for 2:2 cadence detection Other bits are unused. Range: • Lock thresholds = 3–7 • Comb threshold = 4–255 The higher the threshold values, the more stringent the requirements for the deinterlacer: • to mark a pixel as locked and • to start performing reverse telecine deinterlacing You may set lower threshold values for greater sensitivity to cadenced sequences. Intel recommends that you leave all values at their reset value, unless a change to sensitivity is required. Power on value: 0x0010_0707 • Lock thresholds = 7 • Comb threshold = 16
15	Film Unlock Threshold	RW	24	• Bits 2:0 - Unlock threshold for 3:2 cadence detection • Bits 10:8 - Unlock threshold for 2:2 cadence detection • Bits 23:16 - Delta threshold for 2:2 cadence detection Other bits are unused. Range: • Unlock thresholds = 0–5 (must be set to a value lower than the equivalent lock threshold) • Delta threshold = 4–255 The greater the difference between the lock and unlock threshold values, the more stringent the requirements for the deinterlacer: • to mark a pixel as unlocked and • to stop performing inverse telecine deinterlacing You may set a small difference in the threshold values for greater sensitivity to changes in cadenced sequences. Intel recommends that you leave all values to their reset value, unless a change to sensitivity is required. Power on value: 0x0005_0 • Unlock threshold for 3:2 cadence detection = 2 • Unlock threshold for 2:2 cadence detection = 4 • Delta threshold = 5

continued...

Address	Register	RO/RW/WO	Width	Description
16	VOF Lock Delay	RW	5	Specifies the number of fields elapsed after the core detects a cadence, but before reverse telecine begins. The delay allows for any video to drop out. If you set a value less than five, the core locks to cadence quicker but costs potential film artifacts. Range: 0–31 Power on value: 5
17	Minimum Pixels Locked	RW	32	Specifies the least number of pixels showing a cadence for lock to occur. Increase the value of this register if inverse telecine is being erroneously applied to scenes where telecine should not be present. Range: 0–(2^{32}–1) Power on value: 40000 *Note:* Use a higher value for 1080i compared to PAL or NSTC video.
18	Minimum Valid SAD Value	RW	8	When considering whether pixels should remain locked, the SAD values less than this range are ignored. Set this value high to prevent film pixels from decaying over time if they do not show a strong 3:2 cadence. Range: 0–255 Power on value: 255
19	Scene Change Motion Multiplier	RW	8	The Deinterlacer II IP core's scene change detection algorithm detects any scene changes or edits regardless of whether any current cadence continues or is interrupted. Scene changes cause immediate loss and reacquisition of cadence lock, which allows for very smooth deinterlacing of even rapid scene changes. The algorithm detects scene changes based on a set of motion deltas between adjacent fields. The algorithm uses a multiplier in this calculation. This register sets the value of this multiplier, with a default value of 5 corresponding to a 4× motion delta between adjacent scenes. You may set other values as shown in Scene Change Motion Multiplier Value on page 151. Range: 0–9 Power on value: 5
20	Minimum Film to Closed Caption Ratio	RW	32	The Deinterlacer II IP core determines cadence for each pixel based on its immediate surroundings. For some standard definition content, film pixels may drop into video deinterlacing mode due to insufficient cadence signal. When the pixels go into video deinterlacing mode, you may set a minimum film to closed caption ratio. The deinterlacer compares a count of pixels identified as film content in a reference area, with a count of those identified as film content in likely closed caption area. The deinterlacer only enters full video over film mode if the ratio of film content in the reference area to the closed caption area exceeds the threshold value. This register sets the following threshold values:

Minimum Film to Closed Caption Register	Minimum Ratio to Switch into Video Over Film Mode
0	1 (no effect)
1	4
2	16

continued...

<table>
<tr><th>Address</th><th>Register</th><th>RO/RW/WO</th><th>Width</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td>
<table>
<tr><th>Minimum Film to Closed Caption Register</th><th>Minimum Ratio to Switch into Video Over Film Mode</th></tr>
<tr><td>3</td><td>64</td></tr>
<tr><td>4</td><td>256</td></tr>
<tr><td>5</td><td>1024</td></tr>
</table>
Range: 0–5
Power on value: 0</td></tr>
<tr><td>21</td><td>Minimum Pixel Kernel SAD for Field Repeats</td><td>RW</td><td>8</td><td>Once a video achieves cadence lock, every pixel in the frame will either maintain or lose lock independently from then on. If the SAD value is less than the value for this register, then its lock count will be incremented. If it is higher than this value, its lock count will either remain unchanged or be decremented (if less than min valid SAD value).
Range: 0–255
Power on value: 200</td></tr>
<tr><td>22</td><td>History Minimum Value</td><td>RW</td><td>3</td><td>The cadence bias for a given pixel.
Setting a lower value biases the pixels toward film.
Setting a higher value biases the pixels toward video.
The pixel SAD values are scaled according to the recent history that gives the frames an affinity for their historical state.
Range: 0–3
Power on value: 0</td></tr>
<tr><td>23</td><td>History Maximum Value</td><td>RW</td><td>3</td><td>The cadence bias for a given pixel.
Setting a lower value biases the pixels toward film.
Setting a higher value biases the pixels toward video.
The value for this register must be higher than the value for the History Minimum Value register.
Range: 3–7
Power on value: 7</td></tr>
<tr><td>24</td><td>SAD Mask</td><td>RW</td><td>10</td><td>When detecting cadences, the SAD values are AND’ed with this value. This value allows the LSBs to be masked off to provide protection from noise.
For example, use binary 11_1111_0000 to ignore the lower 4 bits of the SAD data when detecting cadences. This register works orthogonally from the Motion Shift register (Offset 25), which affects both motion calculation in general AND cadence detection.
Range: 512–1023
Power on value: 1008 (binary 1111110000)</td></tr>
<tr><td>25</td><td>Motion Shift</td><td>RW</td><td>4</td><td>Specifies the amount of raw motion (SAD) data that is right-shifted. Shifting is used to reduce sensitivity to noise when calculating motion (SAD) data for both bob and weave decisions and cadence detection.
Note: It is very important to set this register correctly for good deinterlacing performance.
Tune this register in conjunction with the motion visualization feature. Higher values decrease sensitivity to noise when calculating motion, but may start to introduce weave artifacts if the value used is too high.</td></tr>
</table>

continued...

Address	Register	RO/RW/WO	Width	Description
				To improve video-over-film mode quality, consider using software to check the `3:2 Cadence State (VOF State)` register, and to add one or two to the motion shift register's value when deinterlacing cadenced content. Range: 0–7 Power on value: 3 Refer to Tuning Motion Shift and Motion Scale Registers on page 152 for more information.
26	`Visualize Film Pixels`	RW	1	Specifies the film pixels in the current field to be colored green for debugging purposes. Use this register in conjunction with the various VOF tuning registers. Range: 0–1 Power on value: 0
27	`Visualize Motion Values`	RW	1	Specifies the motion values for pixels represented with pink for debugging purposes. The greater the luminance of pink, the more motion is detected. Range: 0–1 Power on value: 0
28	Reserved	–	–	This register is reserved for future use.
29	Reserved	–	–	This register is reserved for future use.
30	`Motion Scale`	RW	8	An 8-bit quantity that is used to scale the effect of the detected motion. Refer to Tuning Motion Shift and Motion Scale Registers on page 152 for more information. The register scales the motion according to the following equation: $Scaled\ Motion = Motion . \frac{Motion\ Scale}{32}$ • A value of 32 does not produce any scaling effect. • A value of 1 produces a scaling of 1/32. • A value of 255 produces a scaling of ×7.97 The lower the scaled motion value, the more weave the IP core performs. Therefore, if any weave artifacts are visible, increase this register value. Power on value: 125 (corresponds to ×3.9)

15.13.1. Scene Change Motion Multiplier Value

Table 51. Scene Change Motion Multiplier Value

Scene Change Motion Multiplier Register	Motion in Field Multiplier
0	×1
1	×1.06
2	×1.14
3 (suggested setting for 480i or 576i)	×1.33
4	×2
5 (default and suggested setting for 1080i)	×4
6	×8

continued...

Scene Change Motion Multiplier Register	Motion in Field Multiplier
7	×16
8	×32
9	×64

15.13.2. Tuning Motion Shift and Motion Scale Registers

To tune the motion shift register, follow these steps:

1. Enable motion visualization; set `Visualize Motion Values` register to 1.
2. Enable cadence detection and tuning registers by setting register 12 to 1.
3. Feed the Deinterlacer II IP core with the sequence of interest, ideally one with static areas and areas in motion, such as a waving flag sequence. Areas in the image where motion is detected will appear in pink, with the luminance in proportion to the amount of motion detected.
4. Adjust the `Motion Shift` register through software when the Deinterlacer II IP core runs, to observe the effect on the motion detected. Choose a motion shift value that does not cause any motion to be detected in static areas of the image.
5. When you are satisfied that the correct amount of motion shift is applied, disable motion visualization by resetting the register back to 0.
6. Look for weave artifacts in moving portions of the image, ideally using a test sequence with fast moving sharp edges or narrow lines. If you do not detect any visible weave artifacts, gradually decrease the `Motion Scale` register value from the default 125 until the artifacts become visible.
7. Gradually increase the value of `Motion Scale` register until all the weave artifacts disappear.

16. Frame Buffer II IP Core

The Frame Buffer II IP core buffers video frames into external RAM.

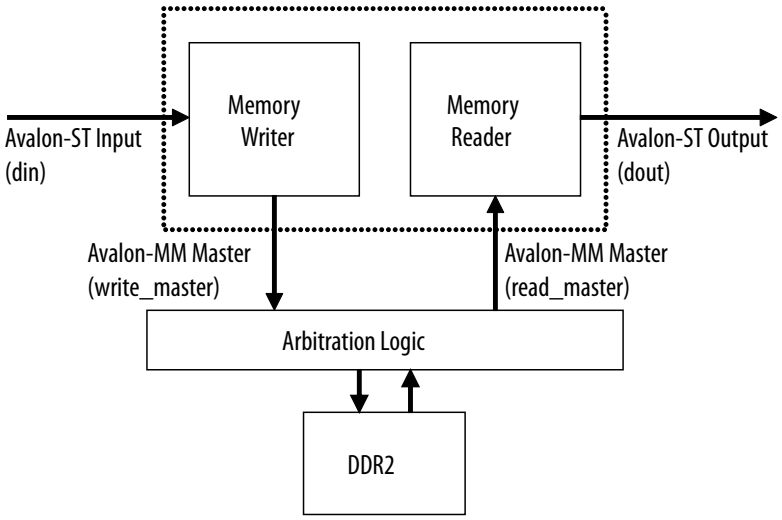
The Frame Buffer II IP core offers the following features:

- Buffers progressive and interlaced video fields.
- Supports double and triple buffering with a range of options for frame dropping and repeating
  - When frame dropping and frame repeating are not allowed—the IP core provides a double-buffering function that can help solve throughput issues in the data path.
  - When frame dropping and/or frame repeating are allowed—the IP core provides a triple-buffering function that can be used to perform simple frame rate conversion.
- Supports up to 8 pixels per transmission.
- Supports a configurable inter-buffer offset to allow the best interleaving of DDR banks for maximum efficiency
- Supports compile-time or run-time controlled variable buffer delay up to 4,095 frames
- Supports reader-only or writer-only modes
- Configurable user packet behavior

The Frame Buffer II IP core has two basic blocks:

- Writer—stores input pixels in memory
- Reader—retrieves video frames from the memory and produces them as outputs

Figure 72. Frame Buffer Block Diagram



16.1. Double Buffering

For double-buffering, the Frame Buffer II IP core uses two frame buffers in external RAM.

- The writer uses one buffer to store input pixels.
- The reader locks the second buffer that reads the output pixels from the memory.
- When both writer and reader complete processing a frame, the buffers are exchanged.
- The input frame can then be read back from the memory and sent to the output, while the buffer that has just been used to create the output can be overwritten with fresh input.

This feature is used when:

- The frame rate is the same both at the input and at the output sides but the pixel rate is highly irregular at one or both sides.
- A frame has to be received or sent in a short period of time compared with the overall frame rate. For example, after the Clipper IP core or before one of the foreground layers of the Alpha Blending Mixer IP core.

16.2. Triple Buffering

For triple-buffering, the IP core uses three frame buffers in external RAM.

- The writer uses one buffer to store input pixels.
- The reader locks the second buffer that reads the output pixels from the memory.
- The third buffer is a spare buffer that allows the input and the output sides to swap buffers asynchronously. The spare buffer can be *clean* or *dirty*.
  - Considered *clean* if it contains a fresh frame that has not been sent.
  - Considered *dirty* if it contains an old frame that has already been sent by the reader component.

When the writer completes storing a frame in memory, it swaps its buffer with the spare buffer if the spare buffer is *dirty*.

- The buffer locked by the writer becomes the new spare buffer and is *clean* because it contains a fresh frame.
- If the spare buffer is already *clean* when the writer completes writing the current input frame:
  - If dropping frames is allowed—the writer drops the newly received frame and overwrites its buffer with the next incoming frame.
  - If dropping frames is not allowed—the writer stalls until the reader completes its frame and replaces the spare buffer with a *dirty* buffer.

When the reader completes reading and produces a frame from memory, it swaps its buffer with the spare buffer if the spare buffer is *clean*.

- The buffer locked by the reader becomes the new spare buffer; and is *dirty* because it contains an old frame that has been sent previously.
- If the spare buffer is already *dirty* when the reader completes the current output frame:
  - If repeating frames is allowed—the reader immediately repeats the frame that has just been sent.
  - If repeating frames is not allowed—the reader stalls until the writer completes its frame and replaces the spare buffer with a *clean* buffer.

16.3. Locked Frame Rate Conversion

The locked frame rate conversion allows the Frame Buffer II IP to synchronize the input and output frame rates through an Avalon memory-mapped slave interface.

The decision to drop and repeat frames for triple-buffering is based on the status of the spare buffer. Because the input and output sides are not tightly synchronized, the behavior of the Frame Buffer II IP is not completely deterministic and can be affected by the *burstiness* of the data in the video system. This may cause undesirable glitches or jerky motion in the video output, especially if the data path contains more than one triple buffer.

By controlling the dropping or repeating behavior, the IP keeps the input and output synchronized. To control the dropping or repeating behavior, you must select triple-buffering mode and turn on **Support for locked frame rate conversion** or **Locked rate support** parameters.

Related Information

Frame Buffer Control Registers on page 159

16.3.1. Converting Frame Rates

You can enable or disable locked frame rate conversion at runtime.

1. Turn on **Locked Rate Support** in the GUI, which adds hardware that supports locked frame rate conversion.
2. Set register 10 with the current input rate and register 11 with the current output rate.

   For more detail, refer to *Frame Buffer II Control Registers Map* table.

For example:

- For a 60Hz frame rate, set a value of 60.
- To convert 60 Hz to 50 Hz, set register 10 to 60 and register 11 to 50.
- To convert from SDI 59.94 to 60, set register 10 to 5994 and register 11 to 6000.

3. Enable or disable locked frame rate conversion with the `Locked Mode Enable` register (register 9).

16.4. Handling of Avalon-ST Video Control Packets and User Packets

The Frame Buffer II IP core stores non-image data packets in memory.

Some applications may repeat and drop the user packets together with their associated frame. For example, if the packets contain frame-specific information such as a frame ID.

The behavior of the IP core is not determined by the field dimensions announced in the Avalon-ST Video control packets and relies exclusively on the `startofpacket` and `endofpacket` signals to delimit the frame boundaries.

- The IP core consequently handles and propagates mislabeled frames. Use this feature in a system where you cannot drop a frame. The latency introduced during the buffering could provide enough time to correct the invalid control packet.
- The buffering and propagation of image data packets incompatible with preceding control packets is an undesired behavior in most systems. Dropping invalid frames is often a convenient and acceptable way of dealing with glitches from the video input.

You can parameterize the Frame Buffer II IP core to drop all mislabeled fields or frames at compile time.

To drop and repeat user packets:

- Set the user packet affinity bit (bit 1) of the Misc. register.
- Turn on **Drop invalid frames** parameter.

Turn on the **Frame repeating** parameter to guarantee that reader keeps on repeating the last valid received frame— freezes the output—when the input drops.

16.5. Frame Buffer Parameter Settings

Table 52. Frame Buffer II Parameter Settings

Parameter	Value	Description
Maximum frame width	32–8192, Default = 1920	Specify the maximum frame width in pixels.
Maximum frame height	32–8192, Default = 1080	Specify the maximum progressive frame height in pixels.

continued...

Parameter	Value	Description
Bits per color sample	4–20, Default = **20**	Select the number of bits per pixel (per color plane).
Number of color planes	1–4, Default = **2**	Select the number of color planes that are sent in sequence.
Color planes transmitted in parallel	**On** or Off	• Turn on to transmit color planes in parallel. • Turn off to transmit color planes in series.
Number of pixels in parallel	**1**, 2, 4, or 8	Specify the number of pixels transmitted or received in parallel.
Interlace support	On or **Off**	Turn on to support consistent dropping and repeating of fields in an interlaced video stream. *Note:* Do not turn on this parameter to double-buffer an interlaced input stream on a field-by-field basis.
Use separate clock for the Avalon-MM master interface(s)	**On** or Off	Turn on to add a separate clock signal for the Avalon-MM master interfaces so that they can run at a different speed to the Avalon-ST processing. This decouples the memory speed from the speed of the data path, and is sometimes necessary to reach performance target.
Avalon-MM master(s) local ports width	16-512, Default = **256**	Specify the width of the Avalon-MM ports used to access external memory.
FIFO depth Write	16–1024, Default = **64**	Select the FIFO depth of the write-only Avalon-MM interface.
Av-MM burst target Write	2–256, Default = **32**	Select the burst target for the write-only Avalon-MM interface.
FIFO depth Read	16–1024, Default = **64**	Select the FIFO depth of the read-only Avalon-MM interface.
Av-MM burst target Read	2–256, Default = **32**	Select the burst target for the read-only Avalon-MM interface.
Align read/write bursts on read boundaries	**On** or Off	Turn on to avoid initiating read and write bursts at a position that would cause the crossing of a memory row boundary.
Maximum ancillary packets per frame	Any 32-bit value, Default = **0**	Specify the number of non-image, non-control, Avalon-ST Video packets that can be buffered with each frame. Older packets are discarded first in case of an overflow. *Note:* The **Maximum length ancillary packets in symbols** parameter is disabled or unused when you specify the number of packets buffered per frame to 0. User packets are no longer delayed through the DDR memory (as with the Frame Buffer I IP core). The packets are instead grouped at the output immediately following the next control packet. Then the video packets swap places with the user packets which arrive before the next control packet.
Maximum length ancillary packets in symbols	10–1024,Default = **10**	Select the maximum packet length as a number of symbols. The minimum value is 10 because this is the size of an Avalon-ST control packet (header included). Extra samples are discarded if the packets are larger than allowed.
Frame buffer memory base address	Any 32-bit value, Default = **0x00000000**	Select a hexadecimal address of the frame buffers in external memory when buffering is used. The information message displays the number of frame buffers and the total memory required at the specified base address.
Enable use of inter-buffer offset	On or **Off**	Turn on if you require maximum DDR efficiency, at the cost of increased memory footprint per frame.
Inter-buffer offset	Any 32-bit value, Default = **0x01000000**	Specify a value greater than the size of an individual frame buffer.

continued...

Parameter	Value	Description
Module is Frame Reader only	On or **Off**	Turn on if you want to configure the frame buffer to be a frame reader.. *Note:* You must select run-time reader control if you select frame reader only.
Module is Frame Writer only	On or **Off**	Turn on if you want to configure the frame buffer to be a frame writer. *Note:* You must select run-time writer control if you select frame writer only.
Frame dropping	On or **Off**	Turn on to allow frame dropping.
Frame repeating	On or **Off**	Turn on to allow frame repetition.
Delay length (frames)	1–2047, Default = **1**	When you turn on the **Drop/repeat user packets** parameters, the IP core implements a minimum of 3 buffers (triple buffer), which gives a delay through the buffer of 1 frame. You can configure the IP core the implement more frame buffers and create a longer delay, up to a maximum of 2047 frames. This feature enables the pausing of a video stream up to 2048 seconds (input frames per second), by applying the back-pressure to the Avalon-ST video output of the frame buffer for the duration of the pause.
Locked rate support	On or **Off**	Turn on to add an Avalon-MM slave interface that synchronizes the input and output frame rates. *Note:* You can only turn on this parameter if you also turn on **Frame dropping**, **Frame repeating**, and **Run-time writer control** parameters.
Drop invalid frames	On or **Off**	Turn on to drop image data packets that have lengths that are not compatible with the dimensions declared in the last control packet.
Drop/repeat user packets	On or **Off**	Turn on to drop or repeat user packets when associated frames are dropped or repeated.
Run-time writer control	On or **Off**	Run-time control for the write interface. The Frame Buffer II has two sides – a reader and a writer. Each side has a register interface, one of which can be configured to be visible to the user. Both control interfaces contain all the necessary registers to control the behavior of the IP core while for the writer, registers 3 and 4 (frame counter and drop/repeat counter) reflect information on dropped frames. *Note:* When you turn on this parameter, the Go bit gets deasserted by default. When you turn off this parameter, the Go is asserted by default. Refer to the *Frame Buffer II Control Register Map*.
Run-time reader control	On or **Off**	Run-time control for the read interface. The Frame Buffer II has two sides – a reader and a writer. Each side has a register interface, one of which can be configured to be visible to the user. Both control interfaces contain all the necessary registers to control the behavior of the IP core while for the reader, registers 3 and 4 (frame counter and drop/repeat counter) reflect information on repeated frames. *Note:* When you turn on this parameter, the Go bit gets deasserted by default. When you turn off this parameter, the Go is asserted by default. Refer to the *Frame Buffer II Control Register Map*.

16.6. Frame Buffer Application Examples

The example use cases provide some guidance for your designs.

Table 53. Example Use Cases for Various Locked and Frame Dropping/Repeating Configurations

Locked Rate Support	Frame Dropping	Frame Repeating	Application Example
Yes	Yes	Yes	A system with source-synchronous input and outputs (sharing the same clock, or *genlocked*), with an input frame rate of 60 Hz and an output frame rate of 24 Hz. The frame buffer implements a triple buffer, providing a regular drop/repeat pattern to ensure that the lower output rate is maintained with minimal perceived jitter in the output video. Register 10 (`Input Frame Rate`) should be set to 60 and register 11 (`Output Frame Rate`) to 24, or any other two *short int* values to represent the 60:24 ratio.
Yes	No	No	Illegal configuration. The frame buffer must be able to drop and repeat frames when input and output rates are locked.
No	Yes	Yes	A system with inputs and outputs which are not source-synchronous (no common clock), with an input frame rate of 60 Hz and an output frame rate of 24 Hz. The frame buffer implements a "triple buffer", providing a variable drop/repeat pattern to accommodate any phase drift seen due to the different clocks. This is the most common configuration used for video standard conversion applications.
No	No	No	A system with source-synchronous input and outputs (sharing the same clock, or *genlocked*), with an input frame rate of 50 Hz and an output frame rate of 50 Hz. This configuration may be useful where the input and output have different burst characteristics, for example a DisplayPort input and an SDI output. The frame buffer implements a "double buffer", providing very little backpressure to the input, while maintaining the required steady rate at the output.

16.7. Frame Buffer Control Registers

A run-time control can be attached to either the writer component or the reader component of the Frame Buffer II IP core but not to both. The width of each register is 32 bits.

Table 54. Frame Buffer II Control Register Map

The table below describes the register map for the Frame Buffer II IP core when configured as a Frame reader only (Reader column), Frame writer only (Writer column) or as a frame buffer (Buffer column). Y indicates the register is applicable for the feature and N/A means not applicable.

Note: Registers 3 and 4 return differently, depending on whether the register interface is a reader or writer control.

Address	Register	Reader	Writer	Buffer	Type	Description
0	`Control`	Y	Y	Y	RW	Bit 0 of this register is the Go bit, Setting this bit to 0 causes the IP core to stop the next time control information is read.

continued...

Address	Register	Reader	Writer	Buffer	Type	Description
						When you enable run-time control, the Go bit gets deasserted by default. If you do not enable run-time control, the Go is asserted by default.
1	Status	Y	Y	Y	RO	Bit 0 of this register is the Status bit, all other bits are unused.
2	Interrupt	Y	Y	Y	RW	The frame writer raises its interrupt line and sets bit 0 of this register when the IP core writes a frame to DDR and the frame is ready to be read. You can clear the interrupt by writing a 1 to this bit. The frame reader raises its interrupt line and sets bit 0 of this register when a complete frame is read from DDR. You can clear the interrupt by writing a 1 to this bit.
3	Frame Counter	Y	Y	Y	RO	For a writer control interface, the counter is incremented if the frame is not dropped. For a reader control interface, this counter is incremented if the frame is not repeated.
4	Drop/Repeat Counter	Y	Y	Y	RO	For a writer control interface, the counter is incremented if the frame is dropped. For a reader control interface, this counter is incremented if the frame is repeated.
5	Frame Information	Y	Y	N/A	RW	• Bit 31 of this register is the Available bit used only in the frame writer mode. A 0 indicates no frame is available and a 1 indicates the frame has been written and available to read. *Note:* In Frame Writer only mode, you must acknowledge each available frame before the next frame is available. Refer to the acknowledge bit in the Misc register. • Bit 30 of this register is unused. • Bits 29 to 26 contain the interlaced bits of the frame last written by the buffer. • Bits 25 to 13 of this register contain the width of the frame last written by the buffer. • Bits 12 to 0 of this register contain the height of the frame last written by the buffer.
6	Frame Start Address	Y	Y	N/A	RW	This register holds the frame start address for the frame last written to DDR by the writer. If configured as a Reader only, you must write the frame start address to this register. For the frame writer configuration, the frame start address is valid only when the Available bit in the Frame Information register is set.
7	Frame Reader	Y	N/A	N/A	RO	• Bit 26 of this register is the Ready bit. This bit is set when the reader is ready to accept the details of the next frame to be read. • Bits 25 to 13 of this register indicate the maximum width of frames that may be read, as configured in the parameter editor. • Bits 12 to 0 of this register indicate the maximum height of frames that may be read, as configured in the parameter editor.

continued...

Address	Register	Reader	Writer	Buffer	Type	Description
8	Misc	Y	Y	Y	RW	• Bit 0 of this register is the acknowledge bit. — Applies only to Frame Writer only mode. — Set this bit to 1 to indicate that the available frame has been completely handled (refer to Available bit in the Frame Information register). — Writing a 1 triggers the buffer to be reset and the Frame Writer reuses the buffer. • Bit 1 of this register is the user packet affinity bit. — Set this bit to 1 you want to drop and repeat user packets together with their associated video packet (this is the next video packet received). This mode allows for specific frame information that must be retained with each frame. — Set this bit to 0 if all user packets are to be produced as outputs in order, regardless of any dropping or repeating of associated video packets. This mode allows for audio or closed caption information. • Bits 15 to 2 of this register are unused. • Bits 27 to 16 of this register contain the frame delay. The default delay value is 1, but you may introduce additional delay to the buffer by writing a value from 2 to 4095 to this register.
9	Locked Mode Enable	N/A	N/A	Y	RW	Bit 0 of this register is enables locked mode. When you set the locked mode bit, the specified Input Frame Rate and Output Frame Rate registers tightly control the dropping and repeating of frames. Setting this bit to 0 switches off the controlled rate conversion and returns the triple-buffering algorithm to a free regime where dropping and repeating is only determined by the status of the spare buffer. Other bits are unused.
10	Input Frame Rate	N/A	N/A	Y	RW	Bits 15:0 contains a short integer value that corresponds to the input frame rate. Other bits are unused.
11	Output Frame Rate	N/A	N/A	Y	RW	Bits 15:0 contains a short integer value that corresponds to the output frame rate. Other bits are unused.

16.7.1. Frame Writer Only Mode

To configure the Frame Buffer II IP core in frame writer mode, select **Module is Frame Writer only** mode in the parameter editor.

In this mode, the frame buffer starts writing incoming video frames to DDR at the Frame buffer memory base address register and automatically advances the write address with each incoming frame. The address of each newly written frame is made available through the Frame Start Address register when the write has completed. This is indicated by the available bit (31) of the Frame Start Address register. This register also holds the height, width, and interlaced information for the written frame. It is not possible to instruct the frame buffer where to write individual frames.

Frame details persist until cleared through a write to bit 0 of the Misc register. The write indicates to the Frame writer that the frame has been completely handled and the buffer may be reused. This also causes the Frame Buffer II to clear the available bit, unless another frame has been received in the meanwhile. In this case, the bit remains set and the new Frame Information becomes available.

The Frame Buffer II also raises its interrupt line and sets bit o of the `Interrupt` register when a new frame is available. The interrupt is cleared down by writing a 1 to the bit.

If additional frames are presented at the input when the frame buffer is already full and you have turned on the **Frame dropping** parameter, the incoming frames will be dropped. If you did not turn on the **Frame dropping** parameter, the Frame Buffer II stalls the input.

16.7.2. Frame Reader Only Mode

To configure the Frame Buffer II IP core in frame reader mode, select **Module is Frame Reader only** mode in the parameter editor.

In this mode, when you set the frame dimensions through the `Frame Information` register and the frame start address through the `Frame Start Address` register, the IP core starts transmitting video frames read from DDR.

Writing to the `Frame Start Address` register initiates the video reading. If new frame dimensions are set, you must perform another write to the `Frame Start Address` register for the new settings to take effect (even if the frame start address is unchanged).

The Frame Buffer II IP core cannot determine if a frame is dirty or clean; the IP core keeps producing a frame from wherever it is currently addressed until a new address is written. Therefore, frame reading applications may use one of the following based on the target application:

- A conventional fixed set of 2 or 3 buffers
- A dozen buffers dynamically allocated at runtime
- A constant test pattern buffer and a set of dynamic buffers

A simple 3-buffer frame reader may operate as follows:

1. Wait until the `ready` bit of the `Frame Reader` register is high, indicating that it is ready to receive details of a new frame to transmit.

   Note: The Frame Buffer II IP core allocates sufficient frame information buffering for the number of frames set through the **Delay Length** parameter.
2. Write the frame dimensions into the `Frame Information` register.

   Note: The frame dimensions include the 4 interlaced bits (e.g. 0 for progressive, 8 for interlaced F0, and 12 for interlaced F1)
3. Write the data for the frame into a buffer area ***N***. This area may be at any address range visible to the frame buffer, as long as the Frame Buffer II is not already transmitting from that region of memory.
4. Write the start address of the buffer to the `Frame Start Address` register (0x6).
5. The Frame Buffer II starts transmitting image data from the buffer.
6. Increment buffer number ***N, N = (N+1)%3***, and repeat from step 1.

The frame reader issues an interrupt immediately after coming out of reset. You can use it to synchronize the system and trigger initiation of the first frame. To use it, you must set up your design to read a new frame after an interrupt indicates the previous one is done.

16.7.3. Memory Map for Frame Reader or Writer Configurations

When creating content for on-screen display using a frame reader, or when processing frame data written to DDR through a frame writer, it is necessary to understand the memory mapping used by the Frame Buffer II IP core.

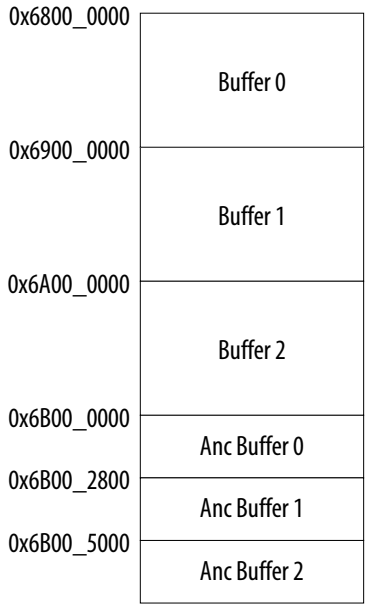
The frame data is tightly packed into memory and aligned on frame (or field) boundaries to minimize storage usage and maximize memory bandwidth usage.

The figure below shows an example of memory map for a frame buffer with the configuration settings below:

- **Bits per pixel per color sample** = 8 bits
- **Number of color planes** = 2
- **Pixels in parallel** = 1
- **Avalon-MM master(s) local ports width** = 25
- **Av-MM burst target Write** = 32
- **Av-MM burst target Read** = 32
- **Align read/write bursts on read boundaries** = On
- **Maximum ancillary packets per frame** = 10
- **Frame buffer memory base address** = 0x6800 0000
- **Enable use of inter-buffer offset** = On
- **Inter-buffer offset** = 0x0100 0000
- **Delay length (frames)** = 1

The maximum length of ancillary packets is ignored if you turn on **Align read/write bursts on read boundaries**.

Figure 73. Example of Memory Map for Base Address 0x6800_0000



The ancillary (user) packets are located in memory after the frame storage when you enable **Align read/write bursts on read boundaries**. Each packet will be offset in memory by (Avalon-MM local ports width * burst target)/8. In this example configuration, the offset is 256*32 / 8 = 1024 (0x400)

Therefore, for the 3 buffers configured, any ancillary packets are written to memory at the following addresses:

Anc buffer 0, anc packet 0 = 0X6B00_0000

Anc buffer 0, anc packet 1 = 0X6B00_0000 + 1*0x400 = 0X6B00_0400

Anc buffer 0, anc packet 2 = 0X6B00_0000 + 2*0x400 = 0X6B00_0800

... Anc buffer 0, anc packet 9 = 0X6B00_0000 + 9*0x400 = 0X6B00_2400

Anc buffer 1, anc packet 0 = 0X6B00_2800

Anc buffer 1, anc packet 1 = 0X6B00_2800 + 1*0x400 = 0X6B00_2800

Anc buffer 1, anc packet 2 = 0X6B00_2800 + 2*0x400 = 0X6B00_2B00...

... Anc buffer 1, anc packet 9 = 0X6B00_2800 + 9*0x400 = 0X6B00_4C00

Anc buffer 2, anc packet 0 = 0X6B00_5000

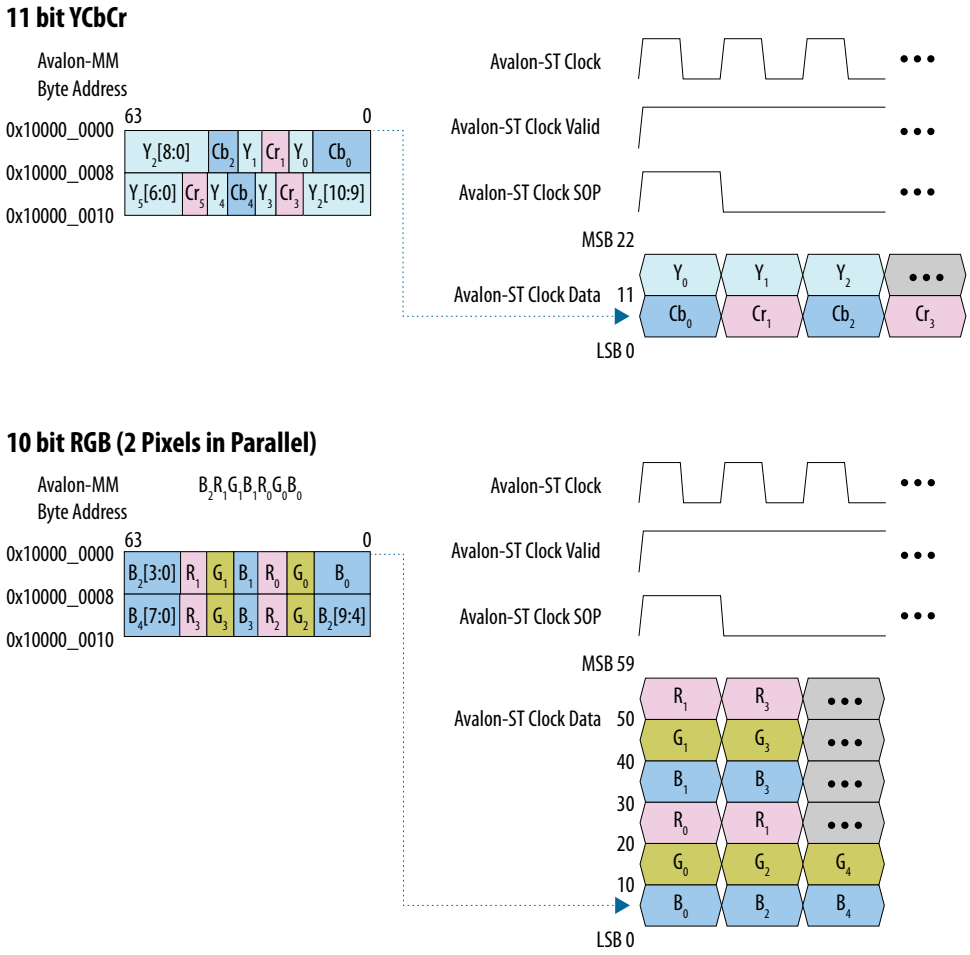
Anc buffer 2, anc packet 1 = 0X6B00_5000 + 1*0x0400 = 0X6B00_5400

Anc buffer 2, anc packet 2 = 0X6B00_5000 + 2*0x0400 = 0X6B00_5800...

... Anc buffer 2, anc packet 9 = 0X6B00_5000 + 9*0x0400 = 0X6B00_7400

Figure 74. Memory Map for Base Address 0x1000_0000 for Non 8-Bit Pixel Values

The figure below illustrates the aliasing that occurs in memory for non 8-bit pixel values that you need to take into account when generating or using pixel addresses in DDR.



The least significant bit (LSB) of the lead pixel is held in the LSB of the first memory word.

17. Gamma Corrector II IP Core

The Gamma Corrector II IP core primarily alters the color values in each pixel in a video stream to correct for the physical properties of the intended display. For example, the brightness displayed by a cathode-ray tube monitor has a nonlinear response to the voltage of a video signal.

The Gamma Corrector II IP core offers the following features:

- Configurable look-up table (LUT) that models the nonlinear function to compensate for the non-linearity.
- Generic LUT based approach, and user programmable LUT contents that allows the IP core to implement any transform that maps individual color plane value at the input to new values at the output according to a fixed mapping.
- Supports up to 4 pixels in parallel.
- Supports extra pipelining registers.

The Gamma Corrector II IP core implements one LUT for each color plane in the pixel. The contents of each LUT are independent of the other LUTs, so each color plane may have its own unique transform mapping. You program the contents of each LUT at run time through an Avalon-MM control slave interface. At this time, the IP core does not support any preset values or a fixed operation mode where you may specify the LUT contents at compile time. As a result, the contents of the LUT(s) are initialized to 0 after every reset. You must overwrite the desired values before processing begins.

You can choose up to two data banks for each LUT to allow two separate transforms to be defined at one time for each color plane. A switch in the register map controls which bank is used to transform the data for each frame. The inclusion of the second LUT bank allows for rapid switching between two transforms on a frame-by-frame basis, and one LUT bank to be updated with a new transform while the video is processed by the other bank without any disruption.

17.1. Gamma Corrector Parameter Settings

Table 55. Gamma Corrector II Parameter Settings

You program the actual gamma corrected intensity values at run time using the Avalon-MM slave interface.

Parameter	Value	Description
Input bits per color sample	4–16, Default = **8**	Select the number of bits per color plane sample for input streams.
Output bits per color sample	4–16, Default = **8**	Select the number of output bits per color plane sample for output streams.
Number of color planes	1–3, Default = **2**	Select the number of color planes per pixel.
Number of pixels in parallel	1, 2, 4, 8, Default = **1**	Select the number of pixels transmitted per clock cycle.

continued...

Parameter	Value	Description
Color planes transmitted in parallel	**On** or Off	Select whether to send the color planes in parallel or in sequence (serially).
Enable 2 banks of LUT coefficients	On or **Off**	Turn on if you want to enable two data banks for each LUT to allow two separate transforms to be defined at one time for each color plane.
How user packets are handled	• No user packets allowed • Discard all user packets received • **Pass all user packets through to the output**	If your design does not require the IP core to propagate user packets, then you may select to discard all user packets to reduce ALM usage. If your design guarantees that the input data stream will never have any user packets, then you further reduce ALM usage by selecting **No user packets allowed**. In this case, the Gamma Corrector II IP core may lock if it encounters a user packet.
Add extra pipelining registers	On or **Off**	Turn on this parameter to add extra pipeline stage registers to the data path. You must turn on this parameter to achieve: • Frequency of 150 MHz for Cyclone V devices • Frequencies above 250 MHz for Intel Arria 10, Intel Cyclone 10 GX, Arria V, or Stratix V devices
Reduced control register readback	On or **Off**	If you do not turn on this parameter, the values written to registers 4 and 5 in the control slave interface can be read back. If you turn on this parameter, the values written to registers 3 and upwards cannot be read back through the control slave interface. This option reduces ALM usage. *Note:* The values of registers 6 and above cannot be read back in any mode.

17.2. Gamma Corrector Control Registers

The Gamma Corrector II IP core requires an Avalon-MM slave interface but the Gamma Corrector IP core can have up to three Avalon-MM slave interfaces.

The Gamma Corrector II IP core requires an Avalon-MM slave interface in all modes to enable run-time updating of the coefficient values. As is the convention with all VIP IP cores, when a control slave interface is included, the IP core resets into a stopped state and must be started by writing a '1' to the Go bit of the control register before any input data is processed.

Table 56. Gamma Corrector II Control Register Map

Address	Register	Description
0	`Control`	Bit 0 of this register is the Go bit, all other bits are unused. Setting this bit to 0 causes the IP core to stop at the end of the next frame/field packet.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused. The IP core sets this address to 0 between frames. The IP core sets this address to 1 when it is processing data and cannot be stopped.
2	`Interrupt`	This bit is not used because the IP core does not generate any interrupts.
3	`Read bank`	• Set to 0 to select LUT bank 0 • Set to 1 to select LUT bank 1 Ignored if dual bank mode is not enabled.
4	`Write bank`	• Set to 0 to enable run-time updating of LUT bank 0 • Set to 1 to enable run-time updating of LUT bank 1

continued...

Address	Register	Description
		Ignored if dual bank mode is not enabled.
5	`Write color plane`	Selects to which color plane (LUT) the writes to the register map will be applied.
6 – $5+2^N$ where *N* is the number of bits per symbol	`LUT contents`	Each register aliases to one address in the selected write color of the selected write bank. *Note:* The values written to registers 6 and above cannot be read back in any mode.

18. Configurable Guard Bands IP Core

The Configurable Guard Bands IP core compares each color plane in the input video stream to upper and lower guard bands values.

If the value in any color plane exceeds the upper guard band then the value is replaced by the upper guard band. Likewise, if the value in any color plane falls below the lower guard band then the value is replaced by the lower guard band.

You may specify different guard band values for each color plane. If you enable the run-time control feature, then these values may be altered at run time through the Avalon-MM control slave interface.

- You may specify the input as unsigned data or signed 2's complement data. In this case, the IP core converts the data to an unsigned format (by adding half the maximum range) before guard banding, and the guard bands are specified as unsigned values.
- You may specify that the output data should be driven as signed data but, as with signed input data, all guard banding is done on the unsigned data before conversions to signed output. The IP core converts the output data to an unsigned format by subtracting half the maximum range after guard banding.
- You may not select both signed input and signed output data.

18.1. Guard Bands Parameter Settings

Table 57. Guard Bands Parameter Settings

Parameter	Value	Description
Bits per color sample	4–16, Default = **8**	Select the number of bits per color plane per pixel.
Number of color planes	1–3, Default = **2**	Select the number of color planes per pixel.
Number of pixels in parallel	1, 2, 4, 8, Default = **1**	Select the number of pixels transmitted per clock cycle.
Color planes transmitted in parallel	**On** or Off	Select whether to send the color planes in parallel or in sequence (serially).
4:2:2 data	On or **Off**	Turn on to indicate that the input data is 4:2:2 sampled. *Note:* 4:2:2 mode does not support odd frame widths and heights.
Signed input data	On or **Off**	Turn on to indicate that the input data should be treated as signed 2's complement numbers.
Signed output data	On or **Off**	Turn on to indicate that the output data should be treated as signed 2's complement numbers
Run-time control	**On** or Off	Turn on to enable run-time control of the guard band values.

continued...

Parameter	Value	Description
		Note: When you turn on this parameter, the Go bit gets deasserted by default. When you turn off this parameter, the Go is asserted by default.
Lower/Upper guard band for color <0–3>	0 to ($1^{\text{Bits per color sample}}$)-1	These parameters to define the guard bands for each color plane (up to 4 colors per pixel—color 0 is in the LSBs of the Avalon-ST Video data bus). If you enable **Run-time control**, these values are just used as defaults at reset and may be overwritten at run time. These are unsigned values.
How user packets are handled	• No user packets allowed • Discard all user packets received • **Pass all user packets through to the output**	If your design does not require the IP core to propagate user packets, then you may select to discard all user packets to reduce ALM usage. If your design guarantees that the input data stream will never have any user packets, then you further reduce ALM usage by selecting **No user packets allowed**. In this case, the IP core may lock if it encounters a user packet.
Add extra pipelining registers	On or **Off**	Turn on this parameter to add extra pipeline stage registers to the data path. You must turn on this parameter to achieve: • Frequency of 150 MHz for Cyclone V devices • Frequencies above 250 MHz for Intel Arria 10, Intel Cyclone 10 GX, Arria V, or Stratix V devices
Reduced control register readback	On or **Off**	If you do not turn on this parameter, the values of all the registers in the control slave interface can be read back after they are written. If you turn on this parameter, you cannot read back the guard band values written through the control slave interface. The control, interrupt and status register values may still be read. This option reduces the size of the control slave logic.

18.2. Configurable Guard Bands Control Registers

Table 58. Configurable Guard Bands Register Map

You may choose to enable an Avalon-MM control slave interface for the Configurable Guard Bands IP core to enable run-time updating of the guard band values. As is the convention with all VIP IP cores, when a control slave interface is included, the IP core resets into a stopped state and must be started by writing a '1' to the Go bit of the control register before any input data is processed.

Address	Register	Description
0	`Control`	Bit 0 of this register is the Go bit. All other bits are unused. Setting this bit to 0 causes the IP core to stop at the end of the next frame/field packet. When you enable run-time control, the Go bit gets deasserted by default. If you do not enable run-time control, the Go is asserted by default.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused. The IP core sets this address to 0 between frames. The IP core sets this address to 1 when it is processing data and cannot be stopped.
2	`Interrupt`	This bit is not used because the IP core does not generate any interrupts.
3	`Lower guard band 0`	Value for lower guard band for color 0.
4	`Upper guard band 0`	Value for upper guard band for color 0.
5	`Lower guard band 1`	Value for lower guard band for color 1.
6	`Upper guard band 1`	Value for upper guard band for color 1.

continued...

Address	Register	Description
7	`Lower guard band 2`	Value for lower guard band for color 2.
8	`Upper guard band 2`	Value for upper guard band for color 2.
9	`Lower guard band 3`	Value for lower guard band for color 3.
10	`Upper guard band 3`	Value for upper guard band for color 3.

19. Interlacer II IP Core

The Interlacer II IP core converts streams of progressive frames into streams of alternating F0 and F1 fields by discarding either the odd lines (F0) or the even lines (F1). The output field rate is consequently equal to the input frame rate.

You can parameterize the Interlacer II IP core to implement a number of optional features:

- Pass through or discard of interlaced fields received at the input.
- Start interlaced output streams created from progressive input with either an F0 or F1 field.
- Override the default alternating F0/F1 output sequence for progressive input frames preceded by control packets with interlaced nibbles indicating that the progressive frame was created by deinterlacing original interlaced content. When you enable this option, the following interlaced nibbles are detected:
  - 0000 and 0100 – progressive frames deinterlaced using F0 as the last field. These are interlaced back into F0 fields.
  - 0001 and 0101 – progressive frames deinterlaced using F1 as the last field. These are interlaced back into F1 fields.

You can also enable an Avalon-MM slave interface to control the behavior of the Interlacer II IP Core at run time. When you enable the Avalon-MM slave interface, you can enable or disable the optional features above at run time. Otherwise, their behavior is fixed by your selection in the parameter editor.

Enabling the Avalon-MM slave interface also allows you to enable and disable all interlacing of progressive frames at run time, giving the option of progressive passthrough. When interlacing progressive input, the interlacer automatically resets to a new F0/F1 sequence when a change of resolution is detected in the incoming control packets, starting again with an F0 or F1 fields as defined by your parameterization or run-time control settings. You may also reset the F0/F1 sequence at any point using the Avalon-MM slave interface (see Interlacer Control Registers on page 173 for details).

19.1. Interlacer Parameter Settings

Table 59. Interlacer II Parameter Settings

Parameter	Value	Description
Maximum image height	32–8192, Default = **1080**	Specify the maximum number of lines in the input frame/field.
Bits per color sample	4–16, Default = **8**	Select the number of bits per color plane per pixel.
Number of color planes	1–3, Default = **2**	Select the number of color planes per pixel.

continued...

Parameter	Value	Description
Number of pixels in parallel	1, 2, 4, 8, Default = **1**	Select the number of pixels transmitted per clock cycle.
Color planes transmitted in parallel	**On** or Off	Select whether to send the color planes in parallel or in sequence (serially).
Enable interlace passthrough	**On** or Off	• Turn on to enable passthrough or interlace fields. • Turn off to discard the interlaced fields at the input. If you enable run- time control, this setting serves as the reset value of this feature which may be turned on or off at run time.
Send F1 first	**On** or Off	• Turn on to start interlaced streams with an F1 field. • Turn off to start interlaced streams with an F0 field. If you enable run- time control, this setting serves as the reset value of this feature which may be turned on or off at run time.
Enable control packet override	**On** or Off	Turn on to enable the control packet nibble to override the default interlace sequence for progressive frames that have been created and tagged by a deinterlacer. If you enable run- time control, this setting serves as the reset value of this feature which may be turned on or off at run time.
Run-time control	On or **Off**	Turn on to enable run-time control of the interlacer features. When you turn on this parameter, the Go bit gets deasserted by default. When you turn off this parameter, the Go is asserted by default. *Note:* The progressive passthrough may only be enabled if you turn on this parameter.
Add extra pipelining registers	On or **Off**	Turn on this parameter to add extra pipeline stage registers to the data path. You must turn on this parameter to achieve: • Frequency of 150 MHz for Cyclone V devices • Frequencies above 250 MHz for Intel Arria 10, Intel Cyclone 10 GX, Arria V, or Stratix V devices
Reduced control register readback	On or **Off**	If you do not turn on this parameter, the values of all the registers in the control slave interface can be read back after they are written. If you turn on this parameter, the values written to registers 3 and upwards cannot be read back through the control slave interface. This option reduces ALM usage.

19.2. Interlacer Control Registers

Table 60. Interlacer II Register Map

You may choose to enable an Avalon-MM control slave interface for the Interlacer II IP core to enable run-time updating of the coefficient values. As is the convention with all VIP IP cores, when a control slave interface is included, the IP core resets into a stopped state and must be started by writing a '1' to the Go bit of the control register before any input data is processed.

Address	Register	Description
0	`Control`	Bit 0 of this register is the Go bit. All other bits are unused. Setting this bit to 0 causes the IP core to stop at the end of the next frame/field packet.

continued...

Address	Register	Description
		When you enable run-time control, the Go bit gets deasserted by default. If you do not enable run-time control, the Go is asserted by default.
1	Status	Bit 0 of this register is the Status bit, all other bits are unused. The IP core sets this address to 0 between frames. The IP core sets this address to 1 when it is processing data and cannot be stopped.
2	Interrupt	This bit is not used because the IP core does not generate any interrupts.
3	Settings	• Bit 0 enables and disables progressive passthrough. • Bit 1 enables and disables interlaced passthrough. • Bit 2 enables and disables control packet interlaced nibble override. • Bit 3 indicates whether the output interlaced sequence should begin with F0 or F1. Set to 0 for F0 and 1 for F1. • Bit 4 allows you to reset the interlacing sequence at run time. To reset the interlaced sequence first stop the IP core using the Go bit in register 0, then write a 1 to bit 4 of this register, and then restart the IP core. All other bits are unused.

20. Scaler II IP Core

The Scaler II IP core resizes video streams, and supports nearest neighbor, bilinear, bicubic and polyphase (with or without simple edge adaptation) scaling algorithms. The Scaler II algorithms support 4:2:2 sampled video data.

The Scaler II IP core automatically changes the input resolution using control packets. You can also configure the IP core to change the output resolution and/or filter coefficients at run time using an Avalon-MM slave interface.

Table 61. Formal definitions of the Scaling Algorithms

Algorithm	Definition
w_{in}	Input image width
h_{in}	Input image height
w_{out}	Output image width
h_{out}	Output image height
F	Function that returns an intensity value for a given point on the input image
O	Function that returns an intensity value on the output image

20.1. Nearest Neighbor Algorithm

Nearest-neighbor algorithm is the lowest quality method, and uses the fewest resources.

If you use the nearest-neighbor algorithm, jagged edges may be visible in the output image because no blending takes place. However, this algorithm requires no DSP blocks, and uses fewer logic elements than the other methods.

Scaling up and down requires one line buffer of the same size as the one line from the clipped input image—taking account of the number of color planes being processed.

For example, scaling up a 100-pixel wide image, which uses 8-bit data with 3 colors in sequence, requires 8 × 3 × 100 = 2,400 bits of memory. Similarly, if the 3 color planes are in parallel, the memory requirement is still 2,400 bits.

For each output pixel, the nearest-neighbor method picks the value of the nearest input pixel to the correct input position. Formally, to find a value for an output pixel located at (i, j), the nearest-neighbor method picks the value of the nearest input pixel to $((i \times w_{in})/w_{out} + 0.5, (j \times h_{in})/h_{out} + 0.5))$.

The 0.5 values in this equation are rounded to the nearest integer input pixel to provide the nearest neighbor pixel.

The calculation performed by the Scaler II is equivalent to the following integer calculation:

$O(i,j) = F((2 \times w_{in} \times i + w_{out})/2 \times w_{out}, (2 \times h_{in} \times j + h_{out})/2 \times h_{out}$

20.2. Bilinear Algorithm

Bilinear algorithm is of higher quality and more expensive than the nearest-neighbor algorithm.

If you use the bilinear algorithm, the jagged edges of the nearest-neighbor method are smoothed out. However, this is at the expense of losing some sharpness on edges.

The bilinear algorithm uses four multipliers per channel in parallel. The size of each multiplier is either the sum of the horizontal and vertical fraction bits plus two, or the input data bit width, whichever is greater. For example, with four horizontal fraction bits, three vertical fraction bits, and eight-bit input data, the multipliers are nine-bit.

With the same configuration but 10-bit input data, the multipliers are 10-bit. The function uses two line buffers. As in nearest-neighbor mode, each of line buffers is the size of a clipped line from the input image. The logic area is more than the nearest-neighbor method.

20.2.1. Bilinear Algorithmic Description

The algorithmic operations of the bilinear method can be modeled using a frame-based method.

To find a value for an output pixel located at (i, j), we first calculate the corresponding location on the input:

$$in_i = (i \times w_{in})/w_{out}$$

$$in_j = (j \times h_{in})/h_{out}$$

The integer solutions $\lfloor in_i \rfloor, \lfloor in_j \rfloor$ to these equations provide the location of the top-left corner of the four input pixels to be summed.

The differences between in_i, in_j, and $\lfloor in_i \rfloor, \lfloor in_j \rfloor$ are a measure of the error in how far the top-left input pixel is from the real-valued position that we want to read from. Call these errors err_i and err_j. The precision of each error variable is determined by the number of fraction bits chosen by the user, Bf_h and Bf_v, respectively.

Their values can be calculated using the following equation:

$$err_i = \frac{((i \times w_{in})\%w_{out}) \times 2^{B_{fh}}}{w_{out}}$$

$$err_j = \frac{((j \times h_{in})\%h_{out}) \times 2^{B_{fv}}}{h_{out}}$$

The sum is then weighted proportionally to these errors.

Note: Because these values are measured from the top-left pixel, the weights for this pixel are one minus the error.

That is, in fixed-point precision: $2^{B_{fh}} - err_i$ and $2^{B_{fv}} - err_j$

The sum is then:

$$O(i,j) \times 2^{B_{fv}+B_{fh}} = F(in_i, in_j) \times (2^{B_{fh}} - err_i) \times (2^{B_{fv}} - err_j) + F(in_i + 1, in_j) \times err_i \times (2^{B_{fv}} - err_j) + F(in_i, in_j + 1) \times (2^{B_{fh}} - err_i) \times err_j + F(in_i + 1, in_j + 1) \times err_i \times err_j$$

20.3. Polyphase and Bicubic Algorithm

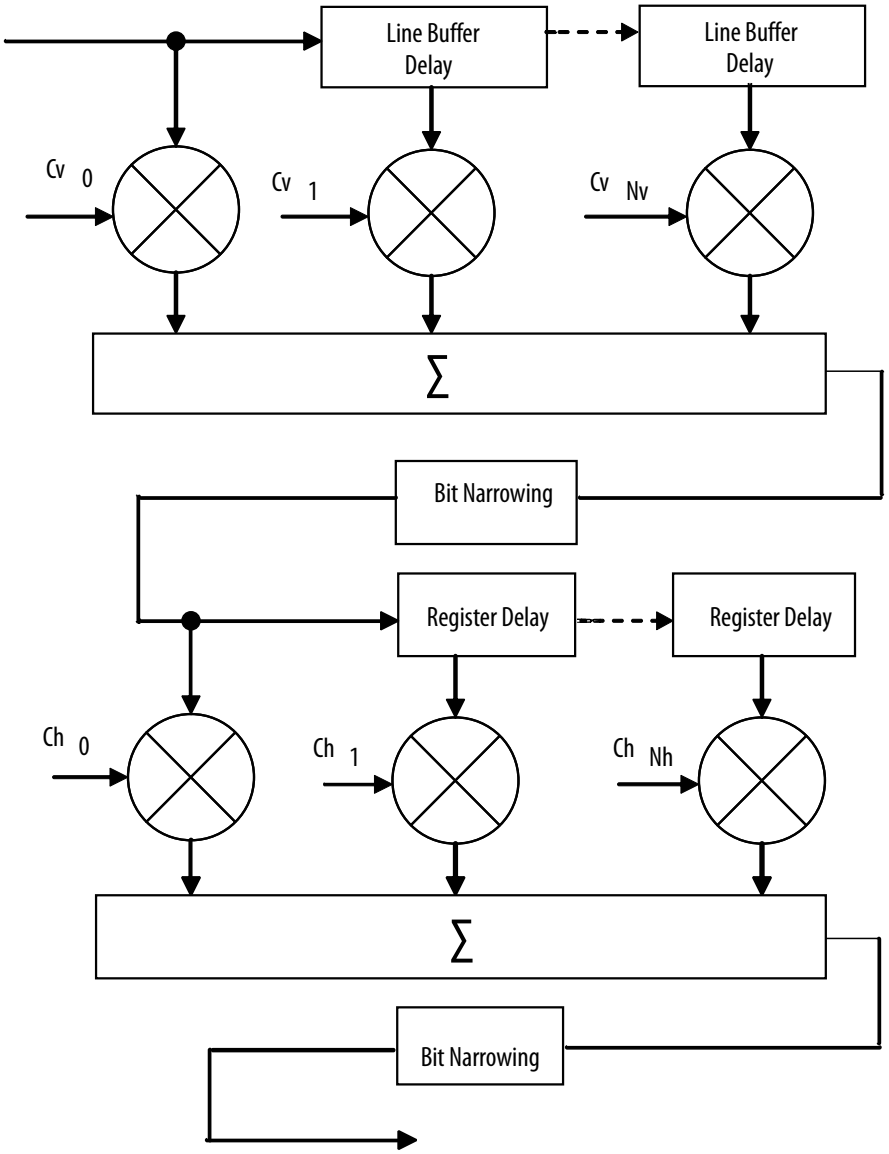
Polyphase and bicubic algorithms offer the best image quality, but use more resources than the other modes of the Scaler II.

The polyphase and bicubic algorithms allow scaling to be performed in such a way as to preserve sharp edges, but without losing the smooth interpolation effect on graduated areas. For down scaling, a long polyphase filter can reduce aliasing effects.

The bicubic and polyphase algorithms use different mathematics to derive their filter coefficients. The implementation of the bicubic algorithm is just the polyphase algorithm with four vertical and four horizontal taps. In the following discussion, all comments relating to the polyphase algorithm are applicable to the bicubic algorithm assuming 4×4 taps.

Figure 75. Polyphase Mode Scaler Block Diagram

The figure below shows the flow of data through an instance of the Scaler II in polyphase mode.



Data from multiple lines of the input image are assembled into line buffers–one for each vertical tap. These data are then fed into parallel multipliers, before summation and possible loss of precision. The results are gathered into registers–one for each horizontal tap. These are again multiplied and summed before precision loss down to the output data bit width.

Note: The progress of data through the taps (line buffer and register delays) and the coefficient values in the multiplication are controlled by logic that is not present in the diagram.

Consider the following for an instance of the polyphase scaler.

- N_v = vertical taps
- N_h = horizontal taps
- B_{data} = bit width of the data samples
- B_v = bit width of the vertical coefficients
- B_h = bit width of the horizontal coefficients
- P_v = user-defined number of vertical phases for each coefficient set (must be a power of 2)
- P_h = user-defined number of horizontal phases for each coefficient set (must be a power of 2)
- C_v = number of vertical coefficient banks
- C_h = number of horizontal coefficient banks

The total number of multipliers is $N_v + N_h$ per channel in parallel.

The width of each vertical multiplier is $max(B_{data}, B_v)$

The width of each horizontal multiplier is the maximum of the horizontal coefficient width, B_h, and the bit width of the horizontal kernel, B_{kh}.

The bit width of the horizontal kernel determines the precision of the results of vertical filtering and is user-configurable.

The memory requirement is N_v line-buffers plus vertical and horizontal coefficient banks. As in the nearest-neighbor and bilinear methods, each line buffer is the same size as one line from the clipped input image.

The vertical coefficient banks are stored in memory that is B_v bits wide and $P_v \times N_v \times C_v$ words deep. The horizontal coefficient banks are stored in memory that is $B_h \times N_h$ bits wide and $P_h \times C_h$ words deep. For each coefficient type, the Intel Quartus Prime software maps these appropriately to physical on-chip RAM or logic elements as constrained by the width and depth requirements.

Note: If the horizontal and vertical coefficients are identical, they are stored in the horizontal memory (as defined above). If you turn on **Share horizontal /vertical coefficients** in the parameter editor, this setting is forced even when the coefficients are loaded at run time.

20.3.1. Double-Buffering

Using multiple coefficient banks allows double-buffering, fast swapping, or direct writing to the scaler's coefficient memories. The IP core specifies the coefficient bank to be read during video data processing and the bank to be written by the Avalon-MM interface separately at run time.

Choosing to have more memory banks allows for each bank to contain coefficients for a specific scaling ratio and for coefficient changes to be accomplished very quickly by changing the read bank. Alternatively, for memory-sensitive applications, use a single bank and coefficient writes have an immediate effect on data processing.

To accomplish double-buffering:

1. Select two memory banks at compile time.
2. At start-up run time, select a bank to write into (for example, 0) and write the coefficients.
3. Set the chosen bank (0) to be the read bank for the Scaler II IP core, and start processing.
4. For subsequent changes, write to the unused bank (1) and swap the read and write banks between frames.

20.3.2. Polyphase Algorithmic Description

The algorithmic operations of the polyphase scaler can be modeled using a frame-based method.

The filtering part of the polyphase scaler works by passing a windowed sinc function over the input data.

- For up scaling, this function performs interpolation.
- For down scaling, it acts as a low-pass filter to remove high-frequency data that would cause aliasing in the smaller output image.

During the filtering process, the mid-point of the sinc function must be at the mid-point of the pixel to output. This is achieved by applying a phase shift to the filtering function.

If a polyphase filter has N_v vertical taps and N_h horizontal taps, the filter is a $N_v \times N_h$ square filter.

Counting the coordinate space of the filter from the top-left corner, (0, 0), the mid-point of the filter lies at $((N_v-1)/2, (N_h-1)/2)$. As in the bilinear case, to produce an output pixel at (i, j), the mid-point of the kernel is placed at $\lfloor in_i \hat{u}, \lfloor in_j \hat{u}$ where in_i and in_j are calculated using the algorithmic description equations. The difference between the real and integer solutions to these equations determines the position of the filter function during scaling.

The filter function is positioned over the real solution by adjusting the function's phase:

$$phase_i = \frac{((i \times w_{in}) \% w_{out}) \times P_h}{w_{out}}$$

$$phase_j = \frac{((j \times h_{in}) \% h_{out}) \times P_v}{h_{out}}$$

The results of the vertical filtering are then found by taking the set of coefficients from $phase_j$ and applying them to each column in the square filter. Each of these N_h results is then divided down to fit in the number of bits chosen for the horizontal kernel. The horizontal kernel is applied to the coefficients from $phase_i$, to produce a single value. This value is then divided down to the output bit width before being written out as a result.

20.3.3. Choosing and Loading Coefficients

The filter coefficients, which the polyphase mode of the scaler uses, may be specified at compile time or at run time.

At compile time, you can select the coefficients from a set of Lanczos-windowed sinc functions, or loaded from a comma-separated variable (CSV) file.

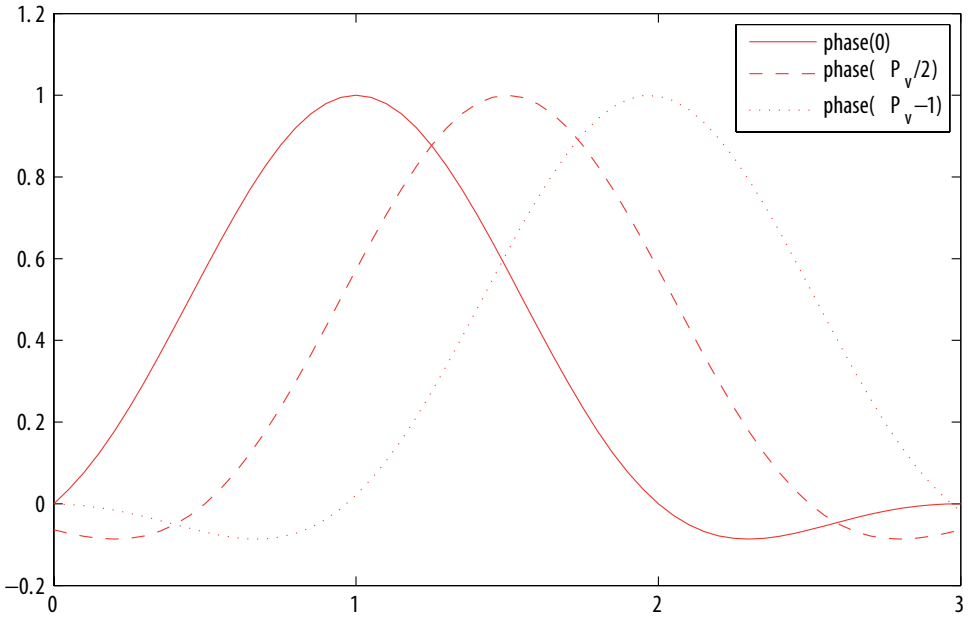
At run time, you specify the coefficients by writing to the Avalon-MM slave control port.

When the coefficients are read at run time, they are checked once per frame and double-buffered so that they can be updated as the IP core processes active data without causing corruption.

Figure 76. Lanczos 2 Function at Various Phases

The figure below shows how a 2-lobe Lanczos-windowed sinc function (usually referred to as Lanczos 2) is sampled for a 4-tap vertical filter.

Note: The two lobes refer to the number of times the function changes direction on each side of the central maxima, including the maxima itself.



The class of Lanczos N functions is defined as:

$$LanczosN(x) = \begin{cases} 1 & x = 0 \\ \frac{\sin(\pi x)}{\pi x}\frac{\sin(\pi x/N)}{\pi x/N} & x \neq 0 \wedge |x| < N \\ 0 & |x| \geq N \end{cases}$$

As can be seen in the figure, phase 0 centers the function over tap 1 on the x-axis. By the equation above, this is the central tap of the filter.

- Further phases move the mid-point of the function in $1/P_V$ increments towards tap 2.
- The filtering coefficients applied in a 4-tap scaler for a particular phase are samples of where the function with that phase crosses 0, 1, 2, 3 on the x-axis.
- The preset filtering functions are always spread over the number of taps given. For example, Lanczos 2 is defined over the range −2 to +2, but with 8 taps the coefficients are shifted and spread to cover 0 to 7.

Compile-time custom coefficients are loaded from a CSV file. One CSV file is specified for vertical coefficients and one for horizontal coefficients. For *N* taps and *P* phases, the file must contain *N*×*P* values. The values must be listed as *N* taps in order for phase 0, *N* taps for phase 1, up to the *N*th tap of the *P*th phase. You are not required to present these values with each phase on a separate line.

The values must be pre-quantized in the range implied by the number of integer, fraction and sign bits specified in the parameter editor, and have their fraction part multiplied out. The sum of any two coefficients in the same phase must also be in the declared range. For example, if there is 1 integer bit, 7 fraction bits, and a sign bit, each value and the sum of any two values must be in the range [−256, 255] representing the range [−2, 1.9921875].

The bicubic method does not use the preceding steps, but instead obtains weights for each of the four taps to sample a cubic function that runs between tap 1 and tap 2 at a position equal to the phase variable described previously. Consequently, the bicubic coefficients are good for up scaling, but not for down scaling.

If the coefficients are symmetric and provided at compile time, then only half the number of phases are stored. For *N* taps and *P* phases, an array, *C*[*P*][*N*], of quantized coefficients is symmetric if for all $p \in [1, P-1]$ and

$$t \in [0, N-1], C[p][t] = C[P-p][N-1-t]$$

.

That is, phase 1 is phase *P*−1 with the taps in reverse order, phase 2 is phase *P*−2 reversed, and so on.

The predefined Lanczos and bicubic coefficient sets satisfy this property. If you select **Symmetric** for a coefficients set in the Scaler II IP core parameter editor, the coefficients will be forced to be symmetric.

20.4. Edge-Adaptive Scaling Algorithm

The edge-adaptive scaling algorithm is almost identical to the polyphase algorithm. It has extensions to detect edges in the input video and uses a different set of scaling coefficients to generate output pixels that lie on detected edges.

In the edge-adaptive mode, each bank of scaling coefficients inside the IP core consists of the following two full coefficient sets:

- A set for pixels that do not lie on the edge—allows you to select a coefficient set with a softer frequency response for the non-edge pixels.
- A set for pixels that lie on the edges—allows you to select a coefficient set with a harsher frequency response for the edge pixels.

These options potentially offer you a more favorable trade-off between blurring and ringing around edges. The Scaler II requires you to select the option to load coefficients at run time to use the edge-adaptive mode; the IP core does not support fixed coefficients set at compile time.

Note: Intel recommends that you use Lanczos-2 coefficients for the non-edge coefficients and Lanczos-3 or Lanczos-4 for the edge coefficients.

20.5. Scaler II Parameter Settings

Table 62. Scaler II Parameter Settings

Parameter	Value	Description
Number of pixels in parallel	**1**, 2, 4, 8	Select the number of pixels transmitted per clock cycle.
Bits per symbol	4–20, Default = **10**	Select the number of bits per color plane.
Symbols in parallel	1–4, Default = **2**	Select the number of color planes sent in parallel.
Symbols in sequence	1–4, Default = **1**	Select the number of color planes that are sent in sequence.
Enable runtime control of output frame size and edge/ blur thresholds	**On** or Off	Turn on to enable run-time control of the output resolution through the Avalon-MM interface. If you turn off this option, the output resolution is fixed at the values you specify for the **Maximum output frame width** and **Maximum output frame height** parameters. *Note:* When you turn on this parameter, the Go bit gets deasserted by default. When you turn off this parameter, the Go is asserted by default.
Maximum input frame width	32–8192, Default = **1920**	Select the maximum width for the input frames (in pixels).
Maximum input frame height	32–8192, Default = **1080**	Select the maximum height for the input frames (in pixels).
Maximum output frame width	32–8192, Default = **1920**	Select the maximum width for the output frames (in pixels).
Maximum output frame height	32–8192, Default = **1080**	Select the maximum height for the output frames (in pixels).
4:2:2 video data	**On** or Off	• Turn on to use the 4:2:2 data format. *Note:* The IP core does not support odd heights or widths in 4:2:2 mode. • Turn off to use the 4:4:4 video format.
No blanking in video	On or **Off**	Turn on if the input video does not contain vertical blanking at its point of conversion to the Avalon-ST video protocol.
Scaling algorithm	• Nearest Neighbor • Bilinear • Bicubic • **Polyphase** • Edge Adaptive	Select the scaling algorithm.

continued...

Parameter	Value	Description
Share horizontal and vertical coefficients	On or **Off**	Turn on to force the bicubic and polyphase algorithms to share the same horizontal and vertical scaling coefficient data.
Vertical filter taps	4–64, Default = **8**	Select the number of vertical filter taps for the bicubic and polyphase algorithms.
Vertical filter phases	1–256, Default = **16**	Select the number of vertical filter phases for the bicubic and polyphase algorithms.
Horizontal filter taps	4–64, Default = **8**	Select the number of horizontal filter taps for the bicubic and polyphase algorithms.
Horizontal filter phases	1–256, Default = **16**	Select the number of horizontal filter phases for the bicubic and polyphase algorithms.
Default edge threshold	0 to $2^{\text{bits per symbol}}-1$, Default = **7**	Specify the default value for the edge-adaptive scaling mode. This value will be the fixed edge threshold value if you do not turn on **Enable run-time control of input/ output frame size** and edge/blur thresholds.
Vertical coefficients signed	**On** or Off	Turn on to force the algorithm to use signed vertical coefficient data.
Vertical coefficient integer bits	0–32, Default = **1**	Select the number of integer bits for each vertical coefficient.
Vertical coefficient fraction bits	1–32, Default = **7**	Select the number of fraction bits for each vertical coefficient.
Horizontal coefficients signed	**On** or Off	Turn on to force the algorithm to use signed horizontal coefficient data.
Horizontal coefficient integer bits	0–32, Default = **1**	Select the number of integer bits for each horizontal coefficient.
Horizontal coefficient fraction bits	1–32, Default = **7**	Select the number of fraction bits for each horizontal coefficient.
Fractional bits preserved	0–32, Default = **0**	Select the number of fractional bits you want to preserve between the horizontal and vertical filtering.
Load scaler coefficients at runtime	On or **Off**	Turn on to update the scaler coefficient data at run time.
Vertical coefficient banks	1–32, Default = **1**	Select the number of banks of vertical filter coefficients for polyphase algorithms.
Vertical coefficient function	• **Lanczos_2** • Lanczos_3 • Custom	Select the function used to generate the vertical scaling coefficients. Select either one for the pre-defined Lanczos functions or choose **Custom** to use the coefficients saved in a custom coefficients file.
Vertical coefficients file	User-specified	When a custom function is selected, you can browse for a comma-separated value file containing custom coefficients. Key in the path for the file that contains these custom coefficients.
Horizontal coefficient banks	1–32, Default = **1**	Select the number of banks of horizontal filter coefficients for polyphase algorithms.
Horizontal coefficient function	• **Lanczos_2** • Lanczos_3 • Custom	Select the function used to generate the horizontal scaling coefficients. Select either one for the pre-defined Lanczos functions or choose **Custom** to use the coefficients saved in a custom coefficients file.
Horizontal coefficients file	User-specified	When a custom function is selected, you can browse for a comma-separated value file containing custom coefficients. Key in the path for the file that contains these custom coefficients.

continued...

Parameter	Value	Description
Add extra pipelining registers	On or **Off**	Turn on to add extra pipeline stage registers to the data path. You must to turn on this option to achieve: • Frequency of 150 MHz for Cyclone IV devices • Frequencies above 250 MHz for Arria II, Stratix IV, or Stratix V devices
Reduced control slave register readback	On or **Off**	If you do not turn on this parameter, the values of all the registers in the control slave interface can be read back after they are written. If you turn on this parameter, the values written to registers 3 and upwards cannot be read back through the control slave interface. This option reduces ALM usage.
How user packets are handled	• No user packets allowed • Discard all user packets received • **Pass all user packets through to the output**	If you design does not require the Scaler II IP core to propagate user packets, then you may select to discard all user packets to reduce ALM usage. If your design guarantees there will never be any user packets in the input data stream, then you further reduce ALM usage by selecting **No user packets allowed**. In this case, the Scaler II IP core may lock if it encounters a user packet.

20.6. Scaler II Control Registers

The control data is read once at the start of each frame and is buffered inside the IP core, so the registers can be safely updated during the processing of a frame.

Table 63. Scaler II Control Register Map

The coefficient bank that is being read by the IP core must not be written to unless the core is in a stopped state. To change the contents of the coefficient bank while the IP core is in a running state, you must use multiple coefficient banks to allow an inactive bank to be changed without affecting the frame currently being processed. The Scaler II IP core allows for dynamic bus sizing on the slave interface. The slave interface includes a 4-bit byte enable signal, and the width of the data on the slave interface is 32 bits.

Note: The N_{taps} is the number of horizontal or vertical filter taps, whichever is larger.

Address	Register	Description
0	`Control`	• Bit 0 of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the IP core to stop the next time control information is read. When you enable run-time control, the `Go` bit gets deasserted by default. If you do not enable run-time control, the `Go` is asserted by default. • Bit 1 enables the edge adaptive coefficient selection—set to 1 to enable this feature.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused. • It is set to 0 if the IP core has not been started. • It is set to 1 while the IP core is processing data and cannot be stopped.
2	`Interrupt`	This bit is not used because the IP core does not generate any interrupts.
3	`Output Width`	The width of the output frames in pixels.
4	`Output Height`	The height of the output frames in pixels.

continued...

Address	Register	Description
5	`Edge Threshold`	Specifies the minimum difference between neighboring pixels beyond which the edge-adaptive algorithm switches to using the edge coefficient set. To get the threshold used internally, this value is multiplied by the number of color planes per pixel.
6	–	Reserved.
7	–	Reserved.
8	`Horizontal Coefficient Write Bank`	Specifies which memory bank horizontal coefficient writes from the Avalon-MM interface are made into.
9	`Horizontal Coefficient Read Bank`	Specifies which memory bank is used for horizontal coefficient reads during data processing.
10	`Vertical Coefficient Write Bank`	Specifies which memory bank vertical coefficient writes from the Avalon-MM interface are made into.
11	`Vertical Coefficient Read Bank`	Specifies which memory bank is used for vertical coefficient reads during data processing.
12	`Horizontal Phase`	Specifies which horizontal phase the coefficient tap data in the `Coefficient Data` register applies to. Writing to this location, commits the writing of coefficient tap data. This write must be made even if the phase value does not change between successive sets of coefficient tap data. To commit to an edge phase, write the horizontal phase number +32768. For example, set bit 15 of the register to 1.
13	`Vertical Phase`	Specifies which vertical phase the coefficient tap data in the `Coefficient Data` register applies to. Writing to this location, commits the writing of coefficient tap data. This write must be made even if the phase value does not change between successive sets of coefficient tap data. To commit to an edge phase, write the vertical phase number +32768. For example, set bit 15 of the register to 1.
14 to 14+N_{taps}	`Coefficient Data`	Specifies values for the coefficients at each tap of a particular horizontal or vertical phase. Write these values first, then the `Horizontal Phase` or `Vertical Phase`, to commit the write.

21. Switch II IP Core

The Switch II IP core enables the connection of input video streams to output video streams.

You can configure the connections at run time through a control input.

The Switch II IP core offers the following features:

- Connects up to twelve input videos to twelve output videos.
- Does not combine streams.
- Each input to the IP core drives multiple outputs.
- Each output is driven by one input.
- Any input can be disabled when not routed to an output.
- Programs each disabled input to be in either stall or consume mode.
  - A stalled input pulls its ready signal low.
  - A consumed input pulls its ready signal high.
- Supports up to 8 pixels per transmission.

The routing configuration of the Switch II IP core is run-time configurable through the use of an Avalon-MM slave control port. You can write to the registers of the control port at anytime but the IP core loads the new values only when it is stopped. Stopping the IP core causes all the input streams to be synchronized at the end of an Avalon-ST Video image packet.

You can load a new configuration in one of the following ways:

- Writing a 0 to the `Go` register, waiting for the `Status` register to read 0 and then writing a 1 to the `Go` register.
- Writing a 1 to the `Output Switch` register performs the same sequence but without the need for user intervention. This the recommended way to load a new configuration.

Note: The status bit in the Switch II IP core's `Status` register is different from the status bit of the other VIP IP cores. The standard status indication has little value for this IP core because it potentially has multiple video streams passing through the core. For this reason, if the status of the video processing is required for synchronization purposes, use the status bit value(s) from the block(s) either upstream or downstream of the Switch II IP core.

21.1. Switch II Parameter Settings

Table 64. Switch II Parameter Settings

Parameter	Value	Description
Bits per color sample	4–20, Default = **8**	Select the number of bits per pixel (per color plane sample).
Number of color planes	1–3, Default = **3**	Select the number of color planes.
Color planes transmitted in parallel	**On** or Off	• Turn on to set colors planes in parallel. • Turn off to set colors planes in sequence.
Number of pixels in parallel	**1**, 2, 4, or 8	Specify the number of pixels transmitted or received in parallel.
Number of inputs	1–12, Default = **2**	Select the number of Avalon-ST video inputs to the IP core.
Number of outputs	1–12, Default = **2**	Select the number of Avalon-ST video outputs from the IP core.

Attention: Intel recommends that you do not create feedback loops using the Switch II IP core. Feedback loops may cause system-level lockups.

21.2. Switch II Control Registers

Table 65. Switch II Control Register Map

The table below describes the control register map for Switch II IP core.

Address	Register	Description
0	`Control`	Bit 0 of this register is the `Go` bit. • Writing a 1 to bit 0 starts the IP core. • Writing a 0 to bit 0 stops the IP core. Bit 1 of this register is the interrupt enable bit. • Setting this bit to 1 enables the switching complete interrupt.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused. • Reading a 1 from bit 0 indicates the IP core is running—video is flowing through it. • Reading a 0 from bit 0 indicates that the IP has stopped running.
2	`Interrupt`	Bit 0 is the interrupt status bit. When bit 0 is asserted, the switching complete interrupt has triggered. Because the Switch II IP core can only change routing configuration at the end of a video frame, this interrupt triggers to indicate that the requested reconfiguration has completed.
3	`Output Switch`	Writing a 1 to bit 0 indicates that the video output streams must be synchronized; and the new values in the output control registers must be loaded.
4	`Dout0 Output Control`	A one-hot value that selects which video input stream must propagate to this output. For example, for a 3-input switch: • 3'b000 = no output • 3'b001 = din_0 • 3'b010 = din_1 • 3'b100 = din_2
5	`Dout1 Output Control`	As `Dout0 Output Control` but for output `dout1`.

continued...

Address	Register	Description
...	...	...
15	`Dout11 Output Control`	As `Dout0 Output Control` but for output `dout11`.
16	`Din Consume Mode Enable`	One bit per input, reset value of 0. • If this bit is set, the associated input is placed in consume mode when it is disabled (not routed to an output). • If this bit is not set, the input is placed in stall mode when disabled. For example, for a 3-input switch: • 3'b000 = All disabled inputs in stall mode • 3'b011 = If disabled, din_2 in stall mode, din_0 and din_1 in consume mode.

22. Test Pattern Generator II IP Core

The Test Pattern Generator II IP core produces an Avalon-ST compliant output stream containing one of five possible fixed test images. The IP core enables you to adjust the output resolution, color space, subsampling, and selected test patterns at run time.You can use this IP core during the design cycle to validate a video system without the possible throughput issues associated with a real video input.

The Test Pattern Generator II IP core offers the following features:

- Supports Avalon-ST Video protocol, including variable color and subsampling interfaces.
- Produces data on request and consequently permits easier debugging of a video data path without the risks of overflow or misconfiguration associated with the use of the Clocked Video Input IP cores or of a custom component using a genuine video input.
- Supports up to 8 pixels per cycle.

22.1. Test Patterns

The Test Pattern Generator II IP core enables you to to choose between 5 different test patterns, either at run time or compile time.

- Color bars
- Grayscale bars
- Black and white bars
- SDI pathological
- Uniform background

22.1.1. Color Bars

In this mode, the Test Pattern Generator II IP core produces a test pattern with 8 vertical bars, each covering approximately 1/8th of each output video line.

The sequence runs through the eight possible on or off combinations of the 3 color components of the RGB color space, starting with a 75% amplitude white.

- Green: on for the first four bars and off for the last four bars
- Red: cycles on and off every two bars
- Blue: cycles on and off every bar

Figure 77. Color Bars Test Pattern with Border Enabled

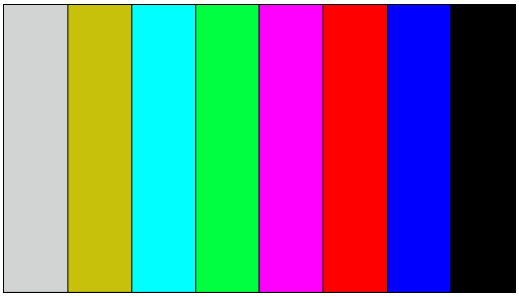


Table 66. Color Values for Each Color Bar Pattern in RGB and YCbCr Color Spaces

The table lists the values used for each color bar in both RGB and YCbCr color spaces for 8 bits per color sample. If the requested output is not in 8 bits per color sample, the IP core truncates or increases these values accordingly.

Color	RGB	YCbCr
White	(180, 180, 180)	(180, 128, 128)
Yellow	(180, 180, 16)	(162, 44, 142)
Cyan	(16, 180, 180)	(131, 156, 44)
Green	(16, 180, 16)	(112, 72, 58)
Magenta	(180, 16, 180)	(84, 184, 198)
Red	(180, 16, 16)	(65, 100, 212)
Blue	(16, 16, 180)	(35, 212, 114)
Black	(16, 16, 16)	(16, 128, 128)

The Test Pattern Generator II IP core can optionally add a black border around the edge of the color bars test pattern. The black border is 2 pixels wide or deep and, if run-time control enabled, you can enable or disable the border at run time.

Each color bar is approximately 1/8th of the width of the output frame, but this width is not exact. The actual width of each color bar is affected by the width and the horizontal subsampling .

- When the output is horizontally subsampled, the pixel-width of each color bar is a multiple of two.
- When the width of the image (excluding the left and right borders) cannot be exactly divided by eight, then the last black bar is larger than the others.

For example, when producing a 640×480 frame in the Y'CbCr color space with 4:2:2 subsampling, the left and right black borders are two pixels wide each, the seven initial color bars are 78 pixels wide ((640–4)/8 truncated down to the nearest multiple of 2) and the final black color bar is 90 pixels wide (640–7×78–4).

22.1.2. Grayscale Bars

The grayscale bars test pattern is similar to the color bars test pattern, except that the bars are gray. In this mode, the Test Pattern Generator II IP core produces a test pattern with color bars of all shades of gray, with the brightness decreasing across the screen from left to right.

As with the color bars test pattern, the black border can be optionally enabled around the edge of the bars, and the width of each bar is calculated in the same way.

Table 67. Color Values for Each Grayscale Bar Pattern in RGB and YCbCr Color Spaces

The table lists the values used for each bar in both RGB and YCbCr color spaces for 8 bits per color sample. If the requested output is not in 8 bits per color sample, the IP core truncates or increases these values accordingly.

Color	RGB	YCbCr
0 (left)	(180, 180, 180)	(180, 128, 128)
1	(162, 162, 162)	(162, 128, 128)
2	(131, 131, 131)	(131, 128, 128)
3	(112, 112, 112)	(112, 128, 128)
4	(84, 84, 84)	(84, 128, 128)
5	(65, 65, 65)	(65, 128, 128)
6	(35, 35, 35)	(35, 128, 128)
7 (right)	(16, 16, 16)	(16, 128, 128)

22.1.3. Black and White Bars

The black and white bars test pattern shares the same properties of the color bars test pattern, except that the bars alternate between black and white. In this mode, the Test Pattern Generator II IP core produces a test pattern with black and white color bars, starting with white.

Table 68. Color Values for Each Bar of the Black and White Bars Pattern in RGB and YCbCr Color Spaces

The table lists the values used for each bar in both RGB and YCbCr color spaces for 8 bits per color sample. If the requested output is not in 8 bits per color sample, the IP core truncates or increases these values accordingly.

Color	RGB	YCbCr
0 (left)	(180, 180, 180)	(180, 128, 128)
1	(16, 16, 16)	(16, 128, 128)
2	(180, 180, 180)	(180, 128, 128)
3	(16, 16, 16)	(16, 128, 128)
4	(180, 180, 180)	(180, 128, 128)
5	(16, 16, 16)	(16, 128, 128)
6	(180, 180, 180)	(180, 128, 128)
7 (right)	(16, 16, 16)	(16, 128, 128)

22.1.4. SDI Pathological

The SDI pathological test pattern is specifically designed to stress test the SDI equalizer and PLL performance.

The test pattern consists of a static test image with the top half of the lines filled with a shade of magenta, and the bottom half of the lines filled with a shade of gray.

Figure 78. SDI Pathological Test Pattern

The option to add a border around the SDI pathological pattern is not enabled because this would reduce the effectiveness of this test for SDI interfaces.

Table 69. Color Values for Each Bar of the SDI Pathological Pattern in RGB and YCbCr Color Spaces

The table lists the values used for each bar in both RGB and YCbCr color spaces for 8 bits per color sample. If the requested output is not in 8 bits per color sample, the IP core truncates or increases these values accordingly.

Color	RGB	YCbCr
Magenta (Upper bar)	(195, 35, 215)	(102, 192, 192)
Gray (Lower bar)	(68, 68, 68)	(68, 128, 128)

22.1.5. Uniform Background

The uniform background test pattern is a complete field or frame of a constant color.

This test pattern has limited value for testing, but you can use the pattern to form a background layer for a mixer. Set the RGB or YCbCr values for the desired color as either fixed, compile-time-set parameters or run-time controlled values through the Avalon-MM control slave interface.

As with the SDI pathological tests pattern, there is no option to add a black border around the edge.

22.2. Output Subsampling and Color Space

The Test Pattern Generator II IP core has parameters that enables you to configure the output subsampling and color space to either be fixed and set at compile time, or to be variable at run time using the Avalon-MM control slave interface.

The **Output Format** parameter sets the subsampling option:

- 4:4:4 – Output is fixed at full sampling for each color plane (can be RGB or YCbCr).
- 4:2:2 – Output is fixed at full sampling on the Y plane and horizontal subsampling on the Cb and Cr planes (RGB data is not supported in 4:2:2 mode).
- 4:2:0 – Output is fixed at full sampling on the Y plane and horizontal and vertical subsampling on the Cb and Cr planes (RGB data is not supported in 4:2:0 mode).
- Monochrome – Output has only one color plane per pixel and represents only a fully sampled Y plane.
- Variable – Output can be configured at run time to be 4:4:4 RGB, 4:4:4 YCbCr, 4:2:2 YCbCr, or 4:2:0 YCbCr.

The Test Pattern Generator II IP core allows you to enable up to 8 different test pattern configurations to switch between at run time.

- Each configuration is a combination of 1 of the 5 available patterns and a formatting option.
- Each test pattern configuration is fixed in one particular output format. However, by selecting between the patterns, you can vary the overall output format at run time.

For example, you may include 4 different configurations, all of which are color bars but with the first configuration set to 4:4:4 RGB, the second set to 4:4:4 YCbCr, the third set to 4:2:2 YCbCr, and the fourth set to 4:2:0 YCbCr. With this setting, you can validate all of the possible configurations of an HDMI 2.0 output.

22.3. Generation of Avalon-ST Video Control Packets and Run-Time Control

The Test Pattern Generator II IP core produces a valid Avalon-ST Video control packet before generating each image data packet , whether it is a progressive frame or an interlaced field.

When the output is interlaced, the Test Pattern Generator II IP cores produces a sequence of pairs of field, starting with:

- F0 if the output is F1 synchronized.
- F1 if the output is F0 synchronized.

When you enable the Avalon slave run-time controller, the resolution of the output can be changed at run-time at a frame boundary, that is, before the first field of a pair when the output is interlaced.

The Test Pattern Generator II IP core does not accept an input stream—so the Avalon-MM slave interface pseudo-code is slightly modified:

```
go = 0;
while (true)
{
        status = 0;
        while (go != 1 )
                wait();
        read_control(); //Copies control to internal register
        status = 1;
do once for progressive output or twice for interlaced output
{
        send_control_packet();
        send_image_data_header();
        output_test_pattern ();
}
}
```

22.4. Test Pattern Generator II Parameter Settings

Table 70. Test Pattern Generator II Parameter Settings

Parameter	Value	Description
Bits per color sample	4-20, Default = **8**	Select the number of bits per pixel (per color sample).
Number of pixels in parallel	**1**, 2, 4, or 8	Select the number of pixels transmitted or received in parallel.
Color planes transmitted in parallel	On or **Off**	This function always produces three color planes but you can select whether they are transmitted in sequence or in parallel. Turn on to transmit color planes in parallel; turn off to transmit color planes in sequence.
Output format	**4:4:4**, 4:2:2, 4:2:0, Variable, or Monochrome	Select the format/sampling rate format for the output frames. *Note:* The IP core does not support odd heights or widths in 4:2:0 mode and odd widths in 4:2:2 mode.
Maximum frame width	32-8192, Default = **1920**	Specify the maximum width of output fields or frames. This parameter is used as the default start-up value only if you turn on run-time control.
Maximum frame height	32-8192, Default = **1080**	Specify the maximum height of output fields or frames. This parameter is used as the default start-up value only if you turn on run-time control.
Default interlacing	• **Progressive output** • Interlaced output (F0 First) • Interlaced output (F1 First)	Specify whether to produce progressive or interlaced video output. For interlaced output, you may select to start with either an F0 field first or an F1 field first. If you turn on run-time control, you can change your selection during run time using the Avalon-MM control slave interface.
Run-time control	On or **Off**	Turn on to enable run-time control of the image size, progressive/interlaced selection, and test pattern selection. *Note:* When you turn on this parameter, the Go bit gets deasserted by default. When you turn off this parameter, the Go asserts by default.
Reduced control register readback	On or **Off**	If you do not turn on this parameter, the values of all the registers in the control slave interface can be read back after they are written. If you turn on this parameter, you won't be able to read back the values written to the register map through the Avalon-MM control slave interface. This option reduces overall resource usage.
Pipeline dout ready	On or **Off**	Turn on to add extra pipeline stage registers to the Avalon-ST ready signals. You must to turn on this option to achieve the frequency of 300 MHz for Intel Arria 10 devices.
Enable border for bar patterns	On or **Off**	Turn to enable the black border on color bar, grayscale bar, and black and white bar test patterns. If you turn on run-time control, you can change your selection during run time using the Avalon-MM control slave interface.
Uniform R or Y	0-255, Default = **16**	Specifies the default color value of the R or Y plane in all uniform background test patterns. If you turn on run-time control, you can change your selection during run time using the Avalon-MM control slave interface.

continued...

Parameter	Value	Description
Uniform G or Cb	0-255, Default = **16**	Specifies the default color value of the G or Cb plane in all uniform background test patterns. If you turn on run-time control, you can change your selection during run time using the Avalon-MM control slave interface.
Uniform B or Cr	0-255, Default = **16**	Specifies the default color value of the B or Cr plane in all uniform background test patterns. If you turn on run-time control, you can change your selection during run time using the Avalon-MM control slave interface.
Number of test patterns	1-8, Default = **1**	Number of test pattern configurations to enable. Run-time control must be enabled for all values other than 1.
Pattern	• **Color bars** • Grayscale bars • Black and White bars • Uniform background • SDI pathological	One parameter for each enabled test pattern configuration. This parameter specifies the test pattern to use for each configuration.
Subsampling & Colorspace	**RGB**, YCbCr 4:4:4, YCbCr 4:2:2, YCbCr 4:2:0, or Monochrome	One parameter for each enabled test pattern configuration. This parameter specifies the subsampling and color space to use for each configuration.

22.5. Test Pattern Generator II Control Registers

The width of each register in the Test Pattern Generator II control register map is 16 bits. The control data is read once at the start of each frame and is buffered inside the IP cores, so that the registers can be safely updated during the processing of a frame or pair of interlaced fields.

After reading the control data, the Test Pattern Generator II IP core generates a control packet that describes the following image data packet. When the output is interlaced, the control data is processed only before the first field of a frame, although a control packet is sent before each field.

Table 71. Test Pattern Generator II Control Register Map

This table describes the control register map for Test Pattern Generator II IP core.

Address	Register	Description
0	`Control`	Bit 0 of this register is the `Go` bit.. Setting this bit to 0 causes the IP core to stop before control information is read. When you enable run-time control, the `Go` bit gets deasserted by default. If you do not enable run-time control, the `Go` is asserted by default. Bit 1 of this register enables or disables the black border in the bars test patterns. This bit gets deasserted (border disabled) at reset. All other bits are unused.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused. The IP core sets this address to 0 between frames. The IP core sets this address to 1 while it is producing data and cannot be stopped.
2	`Interrupt`	Unused.
3	Reserved	Reserved.
4	`Output Width`	The width of the output frames or fields in pixels.

continued...

Address	Register	Description
		Note: Value from 32 up to the maximum specified in the parameter editor.
5	`Output Height`	The progressive height of the output frames or fields in pixels. *Note:* Value from 32 up to the maximum specified in the parameter editor.
6	`Output interlacing`	The output interlacing standard. Set to 0 for progressive, 1 for interlacing with F0 first, and 2 for interlacing with f1 first.
7	`Pattern select`	The test pattern configuration to enable. Write 0 for configuration 0, 1 for configuration 1, and so on.
8	`R/Y value`	The value of the R (or Y) color sample when the test pattern is a uniform color background. *Note:* Available only when the IP core is configured to produce a uniform color background and run-time control interface is enabled.
9	`G/Cb value`	The value of the G (or Cb) color sample when the test pattern is a uniform color background. *Note:* Available only when the IP core is configured to produce a uniform color background and run-time control interface is enabled.
10	`B/Cr value`	The value of the B (or Cr) color sample when the test pattern is a uniform color background. *Note:* Available only when the IP core is configured to produce a uniform color background and run-time control interface is enabled.

23. Trace System IP Core

The Trace System IP core is a debugging and monitoring component for Intel Quartus Prime Standard Edition environments.

The trace system collects data from various monitors, such as the Avalon-ST monitor, and passes it to System Console software on the attached debugging host. System Console software captures and visualizes the behavior of the attached system. You can transfer data to the host over one of the following connections:

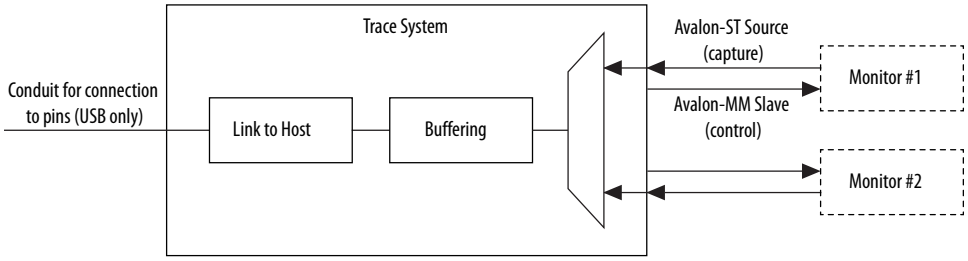
- Direct USB connection with a higher bandwidth; for example On-Board USB-Blaster™ II
  - If you select the USB connection to the host, the trace system exposes the `usb_if` interface.
  - Export this interface from the Platform Designer system and connect to the pins on the device that connects to the On-Board USB-Blaster II.

    Note: To manually connect the `usb_if` conduit, use the USB Debug Link component, located in **Verification ➤ Debug & Performance**.
- JTAG connection
  - If you select the JTAG connection to the host, then the Intel Quartus Prime software automatically makes the pin connection during synthesis.

The Trace System IP core transports messages describing the captured events from the trace monitor components, such as the Frame Reader, to a host computer running the System Console software.

Note: For security reasons, remove the Trace System IP components from designs going to production.

Figure 79. Trace System Functional Block Diagram



When you instantiate the Trace System IP core, turn on the option to select the number of monitors required. The trace system exposes a set of interfaces: `capture`*n* and `control`*n*. You must connect each pair of the interfaces to the appropriate trace monitor component.

The IP core provides access to the control interfaces on the monitors. You can use these control ports to change the capture settings on the monitors; for example, to control the type of information captured by the monitors or to control the maximum data rate sent by the monitor.

Note: Each type of monitor is different. Refer to the relevant documentation of the monitors for more information.

Each trace monitor sends information about interesting events through its capture interface. The trace system multiplexes these data streams together and, if the trace system is running, stores them into a FIFO buffer. The contents of this buffer are streamed to the host using as much as the available trace bandwidth.

The amount of buffering required depends on the amount of jitter inserted by the link, in most cases, the default value of 32 Kbytes is sufficient.

Note: The System Console uses the `sopcinfo` file written by Platform Designer to discover the connections between the Trace System IP core and the monitors. If you instantiate and manually connect the Trace System IP core and the monitors using HDL, the System Console will not detect them.

23.1. Trace System Parameter Settings

Table 72. Trace System Parameter Settings

Parameter	Value	Description
Export interfaces for connection to manual debug fabric	Yes or **No**	If you select **USB** as the connection to host, selecting **Yes** shows the `usb_if` conduit— enables you to manually connect this interface.
Connection to host	• **JTAG** • USB	Select the type of connection to the host running the System Console.
Bit width of capture interface(s)	8–128, Default = **32**	Select the data bus width of the Avalon-ST interface sending the captured information.
Number of inputs	2–16, Default = **2**	Select the number of trace monitors which will be connected to this trace system.
Buffer size	8192–65536, Default = **32768**	Select the size of the jitter buffer in bytes.
Insert pipeline stages	**On** or Off	Turn on to insert the pipeline stages within the trace system. Turning on this parameter gives a higher f_{max} but uses more logic.

23.2. Trace System Signals

Table 73. Trace System Signals

Signal	Direction	Description
`clk_clk`	Input	All signals on the trace system are synchronous to this clock.

continued...

Signal	Direction	Description
		Do not insert clock crossing between the monitor and the trace system components. You must drive the trace monitors' clocks from the same source which drives this signal.
reset_reset	Output	This signal is asserted when the IP core is being reset by the debugging host. Connect this signal to the reset inputs on the trace monitors. Do not reset parts of the system being monitored with this signal because this will interfere with functionality of the system.
usb_if_clk	Input	Clock provided by On-Board USB-Blaster II. All usb_if signals are synchronous to this clock; the trace system provides clock crossing internally.
usb_if_reset_n	Input	Reset driven by On-Board USB-Blaster II.
usb_if_full	Output	Host to the target full signal.
usb_if_empty	Output	Target to the host empty signal.
usb_if_wr_n	Input	Write enable to the host to target FIFO.
usb_if_rd_n	Input	Read enable to the target to host FIFO.
usb_if_oe_n	Input	Output enable for data signals.
usb_if_data	Bidirectional	Shared data bus.
usb_if_scl	Input	Management interface clock.
usb_if_sda	Input	Management interface data.
capture*n*_data	Input	capture*n* port Avalon-ST data bus. This bus enables the transfer of data out of the IP core.
capture*n*_endofpacket	Input	capture*n* port Avalon-ST endofpacket signal. This signal marks the end of an Avalon-ST packet.
capture*n*_empty	Input	capture*n* port Avalon-ST empty signal.
capture*n*_ready	Output	capture*n* port Avalon-ST ready signal. The downstream device asserts this signal when it is able to receive data.
capture*n*_startofpacket	Input	capture*n* port Avalon-ST startofpacket signal. This signal marks the start of an Avalon-ST packet.
capture*n*_valid	Input	capture*n* port Avalon-ST valid signal. The IP core asserts this signal when it produces data.
control*n*_address	Output	control*n* slave port Avalon-MM address bus. This bus specifies a byte address in the Avalon-MM address space.
control*n*_burstcount	Output	control*n* slave port Avalon-MM burstcount signal. This signal specifies the number of transfers in each burst.
control*n*_byteenable	Output	control*n* slave port Avalon-MM byteenable bus.
control*n*_debugaccess	Output	control*n* slave port Avalon-MM debugaccess signal.
control*n*_read	Output	control*n* slave port Avalon-MM read signal. The IP core asserts this signal to indicate read requests from the master to the system interconnect fabric.
control*n*_readdata	Input	control*n* slave port Avalon-MM readdata bus. These input lines carry data for read transfers.

continued...

Signal	Direction	Description
control*n*_readdatavalid	Input	control*n* slave port Avalon-MM readdatavalid signal. The system interconnect fabric asserts this signal when the requested read data has arrived.
control*n*_write	Output	control*n* slave port Avalon-MM write signal. The IP core asserts this signal to indicate write requests from the master to the system interconnect fabric.
control*n*_writedata	Output	control*n* slave port Avalon-MM write signal. The IP core uses these input lines for write transfers.
control*n*_waitrequest	Input	control*n* slave port Avalon-MM waitrequest signal. The system interconnect fabric asserts this signal to cause the master port to wait.

23.3. Operating the Trace System from System Console

System Console provides a GUI and a TCL-scripting API that you can use to control the trace system.

To start System Console, do one of the following steps:

- Run `system-console` from the command line.
- In Platform Designer, on the **Tools** menu, select **Systems Debugging Tools ➤ System Console**.
- In the Intel Quartus Prime software, on the **Tools** menu, select **Transceiver Toolkit**.

  Note: Close the transceiver toolkit panes within System Console.

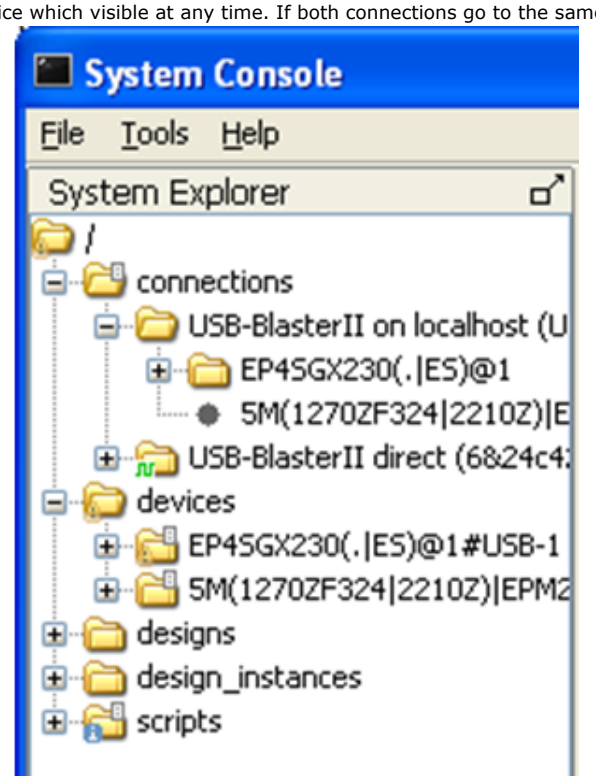
23.3.1. Loading the Project and Connecting to the Hardware

To connect to the Trace System, System Console needs access to the hardware and to the information about what the board does.

To enable access for System Console, follow these steps:

1. Connect to the host.
   - Connect the On-Board USB-Blaster II to the host with the USB cable.
   - Connect the JTAG pins to the host with a USB-Blaster, Ethernet Blaster, or a similar cable.
2. Start System Console and make sure that it detects your device.

This figure shows the System Explorer pane with the `connections` and `devices` folders expanded, with an On-Board USB-Blaster II cable connected. The individual connections appear in the connections folder, in this case the JTAG connection and the direct USB connections provided by the USB-Blaster II. System Console discovers which connections go to the same device and creates a node in the devices folder for each unique device which visible at any time. If both connections go to the same device, then the device only appears once.



3. Load your design into System Console.
   - In the System Console window, on the **File** menu, select **Load Design**. Open the Intel Quartus Prime Project File (`.qpf`) for your design.
   - From the System Console TCL shell, type the following command:

     ```
     [design_load</path/to/project.qpf>]
     ```

   You will get a full list of loaded designs by opening the designs' node within the System Explorer pane on the System Console window, or by typing the following command on the System Console TCL shell:

   ```
   [get_service_paths design]
   ```

4. After loading your design, link it to the devices detected by System Console.
   - In the System Console window, right click on the device folder, and click **Link device to**. Then select your uploaded design. If your design has a JTAG USERCODE, System Console is able to match it to the device and automatically links it after the design is loaded.

     Note: To set a JTAG USERCODE, in the Intel Quartus Prime software, under **Device Assignments** menu, click **Device and Pin Options ➤ General Category**, and turn on **Auto Usercode**.

— From the System Console TCL shell, type the following command to manually link the design:

```
[design_link <design> <device>]
```

Note: Both <design> and <device> are System Console paths as returned by, for example: `[lindex [get_service_paths design] 0]`.

When the design is loaded and linked, the nodes representing the Trace System and the monitors are visible.

23.3.2. Trace Within System Console

When System Console detects a trace system, the Tools menu shows **Trace Table View**. Select this option to display the trace table view configuration dialogue box.

Each detected trace system contains an entry at the **Select hardware** drop down menu. Select one of them and the available settings for its monitors will display. Each type of monitor provides a different set of settings, which can be changed by clicking on the **Value** column.

Figure 80. Trace Control Bar Icons

The figure shows the trace control bar, which lets you control the acquisition of data through the trace system.

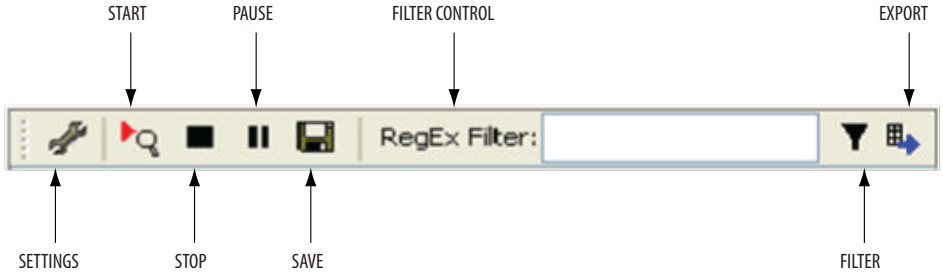


Table 74. Functions of Trace Control Bar Icons

The table lists the trace control bar, which lets you control the acquisition of data through the trace system.

Icon	Function
Settings	Displays the configuration dialog box again.
Start	Tells the trace system to start acquiring data. Data is displayed in the table view as soon as possible after it is acquired.
Stop	Stops acquiring data.
Pause	Stops the display from updating data, but does not affect data acquisition. If you want to examine some data for a length of time, it good to pause so that your data is not aged out of the underlying database.
Save	Saves the raw captured data as a trace database file. You can reload this file using the *Open file* icon in the configuration dialogue.
Filter Control	Lets you filter the captured items to be displayed, but it does not affect acquisition. The filter accepts standard regular expression syntax—you can use filters such as blue/white to select either color.
Filter	Opens the filter settings dialogue, that allows you to select the parts of the captured data you want to display.
Export	Exports a text file containing the current contents of the trace table view. Filters affect the contents of this file.

23.3.3. TCL Shell Commands

You can control the Trace System IP core components from the TCL scripting interface using `trace` service.

Table 75. Trace System Commands

Command	Arguments	Function
`get_service_paths`	`trace`	Returns the System Console names for all the Trace System IP core components which are currently visible.

`claim_service`	`trace`
	`<service_path>`
	`<library_name>`
		Opens a connection to the specified trace service so it can be used. Returns a new path to the opened service.

`close_service`	`trace` `<open_service>`	Closes the service so that its resources can be reused.
`trace_get_monitors`	`<open_service>`	Returns a list of monitor IDs—one for each monitor that is available on this trace system.
`trace_get_monitor_info`	`<open_service>` `<monitor_id>`	Returns a serialized array containing information about the specified monitor. You can use the array set command to convert this into a TCL array.

`trace_read_monitor`	`<open_service>`
	`<monitor_id>`
	`<index>`
		Reads a 32-bit value from configuration space within the specified monitor.

`trace_write_monitor`	`<open_service>`
	`<monitor_id>`
	`<index>`
	`<value>`
		Writes a 32-bit value from configuration space within the specified monitor.

`trace_get_max_db_size`	`<open_service>`	Gets the maximum (in memory) trace database size set for this trace system. If the trace database size exceeds this value, then the oldest values are discarded.
`trace_set_max_db_size`	`<open_service>` `<size>`	Returns the current maximum trace database size. Trace database sizes are approximate but can be used to prevent a high data rate monitor from using up all available memory.
`trace_start`	`<open_service>` `fifo`	Starts capturing with the specified trace system in real time (FIFO) mode.
`trace_stop`	`<open_service>`	Stops capturing with the specified trace system.
`trace_get_status`	`<open_service>`	Returns the current status (idle or running) of the trace system. In future, new status values may be added.
`trace_get_db_size`	`<open_service>`	Returns the approximate size of the database for the specified trace system.
`trace_save`	`<open_service>` `<filename>`	Saves the trace database to disk.
`trace_load`	`<filename>`	Loads a trace database from disk. This returns a new service path, which can be viewed as if it is a trace system. However, at this point, the start, stop and other commands will obviously not work on a file-based node. If you load a new trace database with the `trace_load` command, the trace user interface becomes visible if it was previously hidden.

24. Warp Lite Intel FPGA IP

24.1. Warp Lite IP Release Information

Intel FPGA IP versions match the Intel Quartus Prime Design Suite software versions until v19.1. Starting in Intel Quartus Prime Design Suite software version 19.2, Intel FPGA IP has a new versioning scheme.

The Intel FPGA IP version (X.Y.Z) number can change with each Intel Quartus Prime software version. A change in:

- X indicates a major revision of the IP. If you update the Intel Quartus Prime software, you must regenerate the IP.
- Y indicates the IP includes new features. Regenerate your IP to include these new features.
- Z indicates the IP includes minor changes. Regenerate your IP to include these changes.

Item	Description
IP Version	1.0.0
Intel Quartus Prime	19.4
Release Date	2019.12.16

24.2. About Image Warping

Generally, image warping involves a generic mapping between 2D input and 2D output images. The mapping may be purely arbitrary or may conform to some geometric function. For example, the geometry that describes the distortion introduced by a lens or display system. The required warp mappings are typically: forward and inverse mapping.

Figure 81. Forward Mapping

The figure shows a forward mapping function T{U,V} that maps from the input space (U,V) to the output space (X,Y)

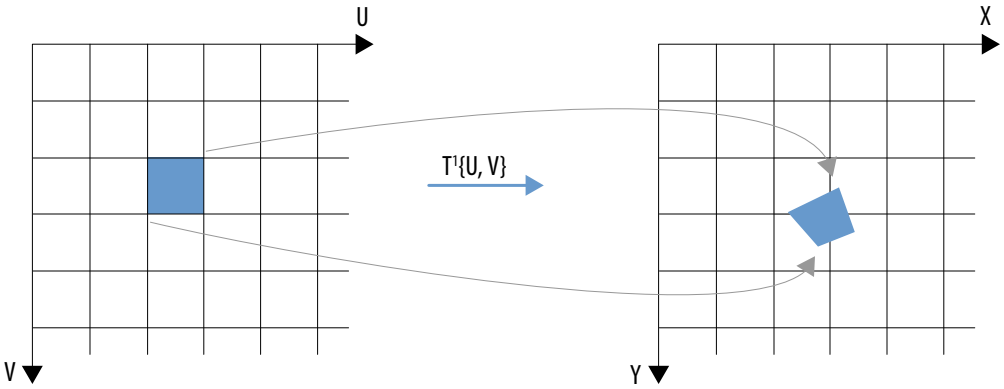
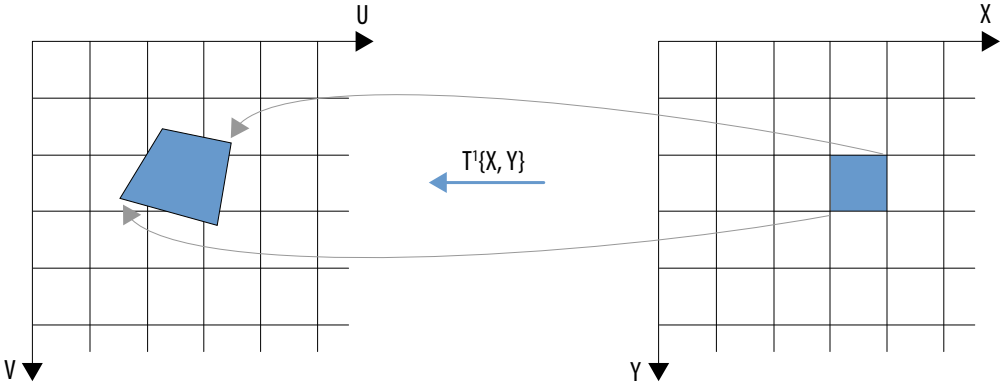


Figure 82. Inverse Mapping

The figure shows the inverse mapping from the output space (X,Y) to the input space (U,V).



24.3. About the Warp Lite Intel FPGA IP

The IP must generate as continuous a flow of data at the output as possible. Therefore, the IP uses inverse mapping, which allows the output pixel stream to be generated directly from the stored input lines. Forward mapping is much more bursty in the mapping of input pixels to output pixels. The IP achieves the implied random access of the inverse mapping by storing the required input pixels in RAM.

The IP implements two warps:

- Vertical keystone (maximum 30% distortion along horizontal edge)
- Horizontal flip

The IP uses a two-dimensional perspective transform to define its warp mapping. The perspective transform is a flexible mapping that you define with eight parameters. The IP supports a minimum resolution of 128x128. Internal logic discards smaller frames. The IP supports a maximum resolution of full HD (1920x1080).

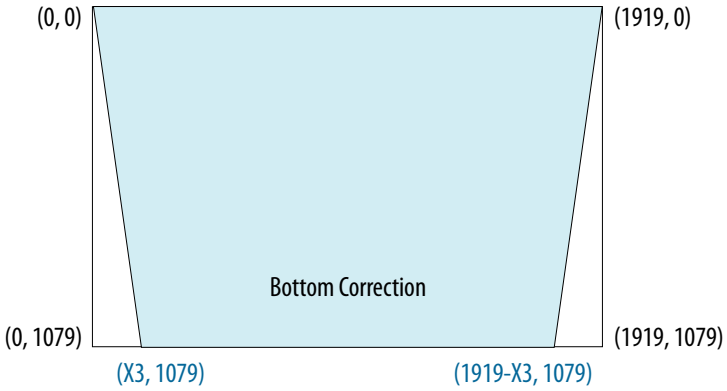
The warp transformation (refer *Warp Definition Equations*) allows for a much wider set of transforms than horizontal flip and vertical keystone. The Warp Lite IP implements a subset of these transforms that it can accommodate using on-chip RAM.

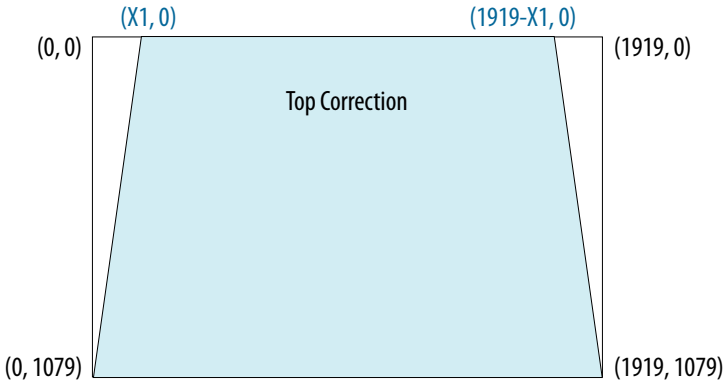
Vertical Keystone Distortion

The IP perspective transform can correct vertical keystone distortions.

Figure 83. Vertical Keystone Region Definition

The figure shows two examples of vertical keystone distortion. The rectangular input image (show in blue) distorts into a quadrilateral that has parallel horizontal sides. The IP can correct for maximum horizontal keystone of 30% distortion. This equates to an upper bound on the X1 and X3 parameters in the output coordinate space of 288 pixels for an image width of 1920 pixels.



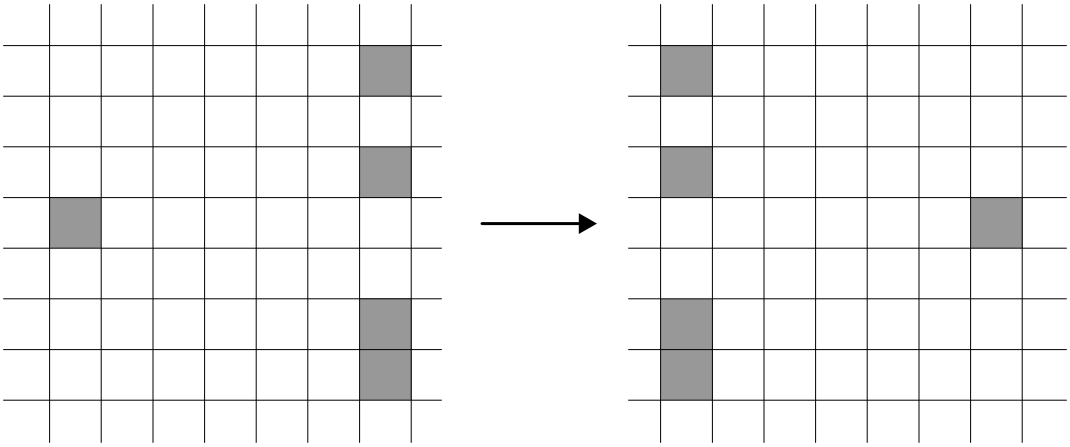


Horizontal Flip

The horizontal flip reverses the image across each line.

The IP uses line stores at its front end. You can read the line data out in reverse order and implement a horizontal flip function.

Figure 84. Horizontal Flip



24.3.1. Warp Definition Equations

To specify the required mapping, the Warp Lite IP uses a perspective transform matrix to define the inverse transformation from the output space (X,Y) to the input space (U,V).

Equation 1. Matrix Form

This matrix applies a mapping that converts the 2-dimensional (X,Y) coordinate space to the 3-dimensional (U', V', W') homogeneous coordinate space.

$$\begin{pmatrix} U' \\ V' \\ W' \end{pmatrix} = \begin{pmatrix} Xscale & Xskew & Xoffset \\ Yskew & Yscale & Yoffset \\ Xpersp & Ypersp & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

Equation 2. Conversion Matrix

To convert to the required 2-dimensional (U, V) space, the IP applies a scaling to the U' and V' coordinates by dividing by the third coordinate W'.

$$\begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} \frac{U'}{W'} \\ \frac{V'}{W'} \end{pmatrix}$$

Based on this matrix form, you control the operation of the Warp IP with these eight values:

- Xscale
- Xskew
- Xoffset
- Yscale
- Yskew
- Yoffset
- Xpersp
- Ypersp

For the vertical keystone mapping, only five of the eight transform parameters: Xscale, Xskew, Xoffset, Yscale and Ypersp. The Xpersp, Yskew and Yoffset are zero.

The run time control register programs each of these values.

The IP defines the vertical keystone distortion within a 1920x1080 screen image with only two parameters: X1 and X3 (refer to figure *Vertical Keystone Region Definition*), the required horizontal offsets in the output coordinate space at the top and bottom of the distorted image respectively.

24.3.1.1. Transform Matrix and Keystone Region Corners

The inverse transformation in the warp definition equations map the four (X,Y) corner points (0,0), (W,0), (0,H) and (W,H) to the corresponding (U,V) points (u1,v1), (u2,v2), (u3,v3) and (u4,v4) respectively.

Applying the transformation of these (X,Y) points and equating them to their corresponding (U,V) points gives the following equations. By solving these equations, the IP determines the transformation matrix values corresponding to the required mapping based on these four corners.

Equation 3. Transform Matrix Equations

$$\begin{bmatrix} \mathrm{W} & 0 & 0 & 0 & -\mathrm{u2*W} & 0 \\ 0 & 0 & \mathrm{W} & 0 & -\mathrm{v2*W} & 0 \\ 0 & H & 0 & 0 & 0 & -u3*H \\ 0 & 0 & 0 & H & 0 & -v3*H \\ W & H & 0 & 0 & -u4*W & -u4*H \\ 0 & 0 & W & H & -v4*W & -v4*H \end{bmatrix} \begin{bmatrix} \mathrm{Xscale} \\ \mathrm{Xskew} \\ \mathrm{Yskew} \\ \mathrm{Yscale} \\ \mathrm{Xpersp} \\ \mathrm{Ypersp} \end{bmatrix} = \begin{bmatrix} u2-u1 \\ v2-v1 \\ u3-u1 \\ v3-v1 \\ u4-u1 \\ v4-v1 \end{bmatrix}$$

$$Xoffset = u1$$

$$Yoffset = v1$$

24.3.1.2. Horizontal Flip Equation

This transform is affine and does not require the **Enable perspective correction** compile time parameter.

This transform requires a minimum requirement of two-line store buffers to operate.

Equation 4. Example matrices format

$$\begin{pmatrix} Xscale & Xskew & Xoffset \\ Yskew & Yscale & Yoffset \\ Xpersp & Ypersp & 1 \end{pmatrix}$$

Figure 85. Horizontal Flip Equation

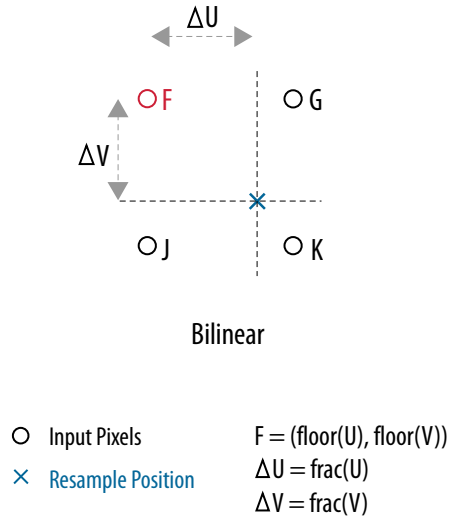
```
-1.00000000 0.00000000 width
0.00000000 1.00000000 0.00000000
0.00000000 0.00000000 1.00000000
```

24.3.2. Resampling and Interpolation

The Warp Lite IP uses an inverse mapping from the output pixel positions to the input pixel positions. Generally, these output positions do not map directly to the available input pixels so the IP implements a resampling and interpolation algorithm. Bilinear interpolation uses the four input pixels surrounding the required input position.

Even though the IP expresses output (X,Y) positions as integers, the corresponding input (U,V) values are fractional because the mappings do not coincide with the exact pixel positions. The pixel position F corresponds to the position (floor(U), floor(V)) The figure shows the fractional offsets (ΔU and ΔV) as the fractional parts of U and V.

Figure 86. Kernel Data Positions Required for Bilinear Interpolation



The IP implements the required interpolation of the input data samples as supplied by the l line store. The IP provides information from the line store for each pixel position in the output image and includes the kernel data and the fractional U and V positions.

24.4. Operating the Warp Lite IP

1. Enable and disable by setting or clearing a control bit in the register map.
2. Generate a set of coefficients for the warp with the `gen_coefficients` API function.
3. Update the IP with the coefficients using the `apply_coefficients` API function.

   Intel recommends using the API rather than setting individual registers. The API checks that the generated coefficients are within IP parameterization limits.

The IP validates that the coefficients are valid for the selected hardware configuration in the software API functions. Invalid sets can cause the IP to wait for conditions that it cannot process.

When you enable the warp IP, it checks arriving control packets to select warp coefficients for the following video frame. If a control packet does not match your coefficients, the warp loads a set of passthrough coefficients. If the resolution changes without coefficients updating, the coefficients might specify thresholds outside the configuration of the IP.

The IP writes the frame to buffers line by line. When the IP receives sufficient lines, the IP generates output lines. It frees lines as it goes, allowing you to write more lines of the incoming frame.

24.5. Warp Lite IP Parameters

The parameters allow you to configure the IP. The Warp Lite IP processes frames of pixel data coming from an Avalon streaming video interface, and outputs the warped frames through an Avalon streaming video output. An Avalon memory-mapped control port allows runtime control of the operation of the IP.

Table 76. Parameters

Parameter	Value	Default	Description
Maximum distortion	5%,10%,15%,20%,25%,30%	15%	Maximum horizontal distortion the warp corrects, which determines the number of lines stores that the IP implements.
Paired with VFB	0	-	When the IP uses a frame buffer, this parameter reduces the number of lines the IP configures for warp.
Maximum image height	128-1080,	1080	Maximum image height in lines.
Maximum image width	128-1920,	1920	Maximum image width in pixels.
Bits per symbol	8 - 16	8	Number of bits per color plane.

24.6. Warp Lite IP Control Registers' Map

Each control register is 32 bits wide.

Table 77. Warp Lite IP Control Registers

Register	Name	Description
0	Control	Bit 0 of this register is the *Go* bit, all other bits are unused. Set this bit to 0, to stop the IP.
1	Status	Bit 0 of this register is the `Status` bit, all other bits are unused. The IP sets it to 1 while it processes data. Clearing the GO bit has no effect until the currently processing frame is completed.
2	Interrupt	Unused. The IP does not generate any interrupts and does not use the register.
3	V Span Error	Reserved.
4	Max V Span	
5	V Span Output Line	
6	Xoffset_fractional	Signed: 1 sign bit; no integer bits; 20 fractional bits
7	Xoffset_integer	Signed: 1 sign bit; 13 integer bits

continued...

The X_offset value is formed by the concatenation of the two registers above.		
8	Yoffset	Reserved.
9	Xscale	Signed: 1 sign bit; 1 integer bit; 20 fractional bits
10	Yscale	Unsigned: 1 integer bit 19 fractional bits
11	Xskew	Signed: 1 sign bit; 1 integer bit; 20 fractional bits
12	Yskew	Reserved..
13	Xpersp	Reserved.
14	Ypersp	Signed: 1sign bit; no integer bits; 29 fractional bits
15	Offscreen R/Y	Pixel value when rendering off screen regions.
16	Offscreen G/Cr	
17	Offscreen B/Cb	
18	Width	The IP calculates the coefficient values for a specific image size. The values are not valid for any other size of image. The `Width` and `Height` registers specify the size of the image for which the IP calculates the coefficients. The IP uses these values to ensure that the coefficients are only applied to images of the correct size.
19	Height	

Off-screen Regions

The inverse mapping from the output coordinate space to the input coordinate space may select regions outside the input frame dimensions. The IP sets the output pixel values for these regions to the values in the offscreen control registers.

Related Information

Warp Lite IP API on page 213

24.7. Warp Lite IP Line Store

The Warp Lite IP uses an on-chip RAM line store to provide the random access required when applying the inverse mapping from the output space to the input space. The maximum distortion, maximum line width, and number of bits per symbol determine the size of this line store.

The line store stores a fixed number of input lines at any one time. The IP configures the line store as dual-port RAM. The number of lines depends on the maximum distortion and the maximum resolution width. You select the required distortion from the configuration parameters.

Table 78. Line Buffer Requirement and M20K Usage for Max Correction Factors (1080p)

The table shows the number of lines generated

Percentage Distortion	Not paired with VFB		Paired with VFB	
	Line Buffers Required	M20K Required	Line Buffers Required	M20K Required
5	16	64	12	48
10	32	128	23	92
15	48	192	35	140

continued...

Percentage Distortion	Not paired with VFB		Paired with VFB	
	Line Buffers Required	M20K Required	Line Buffers Required	M20K Required
20	64	256	47	188
25	81	324	60	240
30	100	400	74	296

24.7.1. Line Store RAM Utilization

Table 79. M20K Availability for Intel Arria 10 Devices

Device	M20K
A10 GX160	440
A10 GX320	891
A10 GX480	1,431
A10 GX570	1,800
A10 GX660	2,131
A10 GX900	2,423
A10 GX1150	2,713

24.7.2. Buffer Ingress and Egress Overlap

The Warp Lite IP line store introduces multiline latency between ingress of input lines and production of output lines.

This latency extends beyond the end of one input frame to the beginning of the next, if the vertical blanking period is shorter than the line store's latency. The IP cannot stall input data at this point. Therefore, the line store allows storage of input lines from one frame to overlap with the reading of output data for the previous frame.

Therefore, you may issue a stop before frame A is complete and not see the IP stop until frame B is output. The IP starts storing frame B before the IP finishes producing frame A. The IP is producing frame B is when the IP issues the stop and must also produce it.

24.8. Warp Lite IP API

The API simplifies programming the control registers.

The recommended sequence at startup is:

```
set_off_screen_R(<unsigned int> value);
set_off_screen_G(<unsigned int> value);
set_off_screen_B(<unsigned int> value);
gen_coefficients(<float> distortion, <bool> top_correction, <bool> h_flip, <int>
width, <int> height)
apply_coefficients()
start()
```

The `start()` function (and its `stop()` counterpart) are inherited from the `VipCore` class.

The sequence to change coefficients in use is simply

```
gen_coefficients(<float> distortion, <bool> top_correction, <bool> h_flip, <int>
width, <int> height)
apply_coefficients()
```

Table 80. Warp IP API Functions

Name	Description
`gen_coefficients`	Generates a valid set of coefficients for the control registers from the input arguments. Write the arguments to the control registers using the apply_coefficients() function. If the input arguments generate an invalid coefficient set, the coefficient set is left unmodified and a terminal message is displayed. If you call the `apply_coefficients()` function, the IP applies the last valid set and you see no change in IP behavior.
`apply_coefficients`	Produces a coefficient set generated by the `gen_coefficients()` function to the control registers. If you never call the `gen_coefficients()` function, the IP produces the default passthrough coefficients. If you calll the `gen_coefficients()` function but it fails, the IP produces the previous valid values. If you set no such values, the IP produces the default passthrough coefficients.
`set_offscreen_R`	Sets the value of the RGB[R]/YCbCr[Y] component for the IP to use for offscreen pixels.
set_offscreen_G	Sets the value of the RGB[G]/YCbCr[Cb] component for the IP to use for offscreen pixels.
set_offscreen_B	Sets the value of the RGB[B]/YCbCr[Cr] component for the IP to use for offscreen pixels.
Start	Sets the GO bit of the control register.
Stop	Clears the GO bit of the control register.

`gen_coefficients` Function

Arguments

`float distortion_ratio`: Amount of distortion to apply from 0 to 0.3

`bool top_correction`: Select edge to which vertical keystone applies. Top(true) or bottom (false).

`bool h_flip`: Select whether to apply horizontal flip (true) or not (false)

`int width`: Width of image to warp. `int height`: Height of image to warp.

Return Value Void

Example `gen_coefficients(0.15, true, false, 1920,1080);` Specify a top edge keystone correction of 15% to HD frames

`gen_coefficients(0,false,true,720,576);`Apply a horizontal flip without keystone to SD frames.

`gen_coefficients(0.3,false,true,1280,720);` Apply a bottom edge keystone of 30% with horizontal flip to 1280x720 frames.

`apply_coefficients` Function

Arguments Void

Return Value Void

Example `apply_coefficients();`

`set_offscreen_R` Function

Arguments `Int`: Value

Return Value Void

Example `Set_offscreen_R(255);` Set the RGB R component to full red for 8 bit color planes.

`set_offscreen_G` Function

Arguments `Int`: Value

Return Value Void

Example `Set_offscreen_G(255);` Set the RGB G component to full green for 8 bit color planes.

`set_offscreen_B` Function

Arguments `Int`: Value

Return Value Void

Example `Set_offscreen_B(255);` Set the RGB B component to full blue for 8 bit color planes.

`Start` Function

Arguments Void

Return Value Void

Example `stop();`

`Start` Function

Arguments	Void
Return Value	Void
Example	`stop();`

25. Avalon-ST Video Stream Cleaner IP Core

The Avalon-ST Video Stream Cleaner IP core removes and repairs the non-ideal sequences and error cases present in the incoming data stream to produce an output stream that complies with the implicit ideal use model.

You can configure the Avalon-ST Stream Cleaner IP core to:

- Remove frames with control packet dimensions not within the specified minimum or maximum values.
- Remove any interlaced fields with more than 540 lines, for example any interlaced content more than 1080i (which the deinterlacer cannot handle).
- Clip input video lines so that the output video line length is an even multiple of a fixed modulo check value. This feature is useful for pipelines with multiple pixels transmitted in parallel to avoid any resolutions that generate a non-zero value on the Avalon-ST empty signal.

If you choose to write your own Avalon-ST Video compliant cores, you may consider adding the Avalon-ST Video Stream Cleaner to the pipeline at the input to your cores. The Avalon-ST Video Stream Cleaner IP core allows you to write the code for your cores without considering their behavior in all the potential error cases.

25.1. Avalon-ST Video Protocol

The Avalon-ST Video protocol uses an implicit model for ideal use conditions.

In the implicit model ideal use conditions, the data stream contains repeating sequences of these set of packets:

N user packets (N ≥ 0) ➤ 1 valid control packet ➤ 1 frame/field packet (matching the dimensions specified in the control packet)

However, the Avalon-ST Video protocol allows for different sequences of input packets without any errors in the data processing. For example:

- Every frame or field packets could receive multiple control packets.
- Some or all of the user packets could arrive between the control and frame or field packet.
- The video need not send a control packet with every frame or field packet if the input resolution and interlace field remains constant for each frame packet.

The Avalon-ST Video protocol also allows for a wide range of error cases that may disrupt the data processing and produce corrupted output video. These error cases include:

- Control packets have insufficient number of beats of data to be decoded
- Length of frame or field packet does not match the resolution specified by the preceding control packet.

25.2. Repairing Non-Ideal and Error Cases

The Avalon-ST Video Stream Cleaner IP core repairs various non-ideal and error cases.

Table 81. Repairing Non-Ideal and Error Cases

This table shows how the Avalon-ST Video Stream Cleaner IP core repairs the non-ideal and error cases.

Non-Ideal/Error Cases	Action
Multiple control packets per frame/field	Uses the values from the final control packet in the sequence to generate a single control packet at the output.
Broken control packets	Ignores and removes the control packets with too few beats of data to decode correctly from the data stream.
User packets between the control and frame/field packet	• Generates a single control packet at the output. • Locates the single control packet between the last user packet (if there are any) and the frame/field packet.
Missing control packet	Removes any frame/field packets without an associated decodable control packet (since the preceding frame) from the stream.
Frame/field dimensions larger than parameterized maxima	If the height or width specified by the preceding control packet is larger than the maximum value set in the parameter editor, the IP core removes the frame/field packet from the stream.
Frame/field dimensions smaller than parameterized minima	If the height or width specified by the preceding control packet is smaller than maximum value set in the parameter editor, the IP core removes the frame/field packet from the stream.
Frame/field packet longer than specified by the control packet	If the frame/field packet contains more beats of data than the beats specified by the height and width values in the preceding control packet, the IP core clips the frame/field packet so its length matches the control packet values.
Frame/field packet shorter than specified by the control packet	If the frame/field packet contains fewer beats of data than the beats specified by the height and width values in the preceding control packet, the IP core pads the frame/field packet with gray pixel data so its length matches the control packet values.
Interlaced data above 1080i (optional)	This optional featured (parameter controlled) will remove any fields with preceding control packets that specify interlaced data greater than 1920 pixels wide or 540 lines per field (greater than 1080i).
Non-modulo line lengths	If you set a modulo check value (set to 1), the Avalon-ST Video Stream Cleaner IP core will check whether the frame/field width for the incoming control packet and the frame/field packets is an integer multiple of this value. If the width is not an integer multiple of the modulo check value, the IP core adjusts the control packet width to the nearest integer multiple. The IP core clips each line in the frame/field packet accordingly.

25.3. Avalon-ST Video Stream Cleaner Parameter Settings

Table 82. Avalon-ST Video Stream Cleaner Parameter Settings

Parameter	Value	Description
Bits per color sample	4–20, Default = **8**	Select the number of bits per color plane.
Number of color planes	1–4, Default = **2**	Select the number of color planes per pixel.
Color planes transmitted in parallel	**On** or Off	Select whether to send the color planes in parallel or sequence (serially).
Number of pixels in parallel	**1**, 2, 4, 8	Select the number of pixels transmitted per clock cycle.
Maximum frame width	32–8192, Default = **1920**	Specify the maximum frame width allowed by the core. The Avalon-ST Video Stream Cleaner removes any frames above the specified width from the data stream.
Maximum frame height	32–8192, Default = **1080**	Specify the maximum frame height allowed by the core. The Avalon-ST Video Stream Cleaner removes any frames above the specified height from the data stream.
How user packets are handled	• Discard all user packets received • **Pass all user packets through to the output**	If you design does not require the Clipper II IP core to propagate user packets, then you may select to discard all user packets to reduce ALM usage. If your design guarantees there will never be any user packets in the input data stream, then you further reduce ALM usage by selecting **No user packets allowed**. In this case, the Clipper II IP core may lock if it encounters a user packet.
Minimum frame width	32–8192, Default = **32**	Specify the minimum frame width allowed by the core. The Avalon-ST Video Stream Cleaner removes any frames below the specified width from the data stream.
Minimum frame height	32–8192, Default = **32**	Specify the minimum frame width allowed by the core. The Avalon-ST Video Stream Cleaner removes any frames below the specified height from the data stream.
Enable control slave port	On or **Off**	Turn on to enable an Avalon-MM control slave port where the error count values can be read and reset.
Only clip even numbers of pixels from the left side	On or **Off**	Frames with widths that are non-integer multiples of the width modulo check value are clipped to the nearest integer multiple. They are clipped as equally as possible on the left and right edges of the image. Turning on this parameter forces the clip on the left edge of the image to be an even number of pixels. The even number is necessary to prevent color swap for 4:2:2 formatted data.
Width modulo check value	**1**, 2, 4, 8, 16, or 32	Specify the width modulo check value.
Remove interlaced fields larger than 1080i	On or **Off**	Turn on to remove interlaced field packets larger than 1080i
Register Avalon-ST ready signals	On or **Off**	Turn on to add extra pipeline stage registers to the data path. You must to turn on this option to achieve: • Frequency of 150 MHz for Cyclone III or Cyclone IV devices • Frequencies above 250 MHz for Arria II, Stratix IV, or Stratix V devices

25.4. Avalon-ST Video Stream Cleaner Control Registers

You may choose to enable an Avalon-MM control slave interface for the Avalon-ST Video Stream Cleaner IP core.

Table 83. Avalon-ST Video Stream Cleaner Control Registers

The table below describes the control register map that controls the Avalon-ST Video Stream Cleaner IP core. Internally the IP core tracks the number of times that various error conditions the IP core encounters and repaired or removed. You can use the control slave interface to read and reset these values.

Address	Register	Description
0	`Control`	Bit 0 of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the Avalon-ST Video Stream Cleaner IP core to stop at the end of the next frame or field packet.
1	`Status`	Bit 0 of this register is the `Status` bit, all other bits are unused. The IP core sets this address to 0 between frames. It is set to 1 while the IP core is processing data and cannot be stopped.
2	`Interrupt`	This bit is not used because the IP core does not generate any interrupts.
3	`Non modulo width count`	Counts the number of frames with widths that are non-integer multiples of the modulo width check value.
4	`Width too small count`	Counts the number of frames with preceding control packets with widths smaller than the value you set for the **Minimum frame width** parameter.
5	`Width too big count`	Counts the number of frames with preceding control packets with widths greater than the value you set for the **Maximum frame width** parameter.
6	`Height too small count`	Counts the number of frames with preceding control packets with heights smaller than the value you set for the **Minimum frame height** parameter.
7	`Height too big count`	Counts the number of frames with preceding control packets with heights greater than the value you set for the **Maximum frame height** parameter.
8	`No valid control packet count`	Counts the number of frames with no valid preceding control packet.
9	`Interlaced greater than 1080i count`	Counts the number of fields with content greater than 1080i.
10	`Mismatch pad frame count`	Counts the number of frame packets that have been padded to match the length implied by the control packet.
11	`Mismatch crop frame count`	Counts the number of frame packets that have been cropped to match the length implied by the control packet.
12	`Counter reset`	Writing any value to this register will reset all error count values to 0.

26. Avalon-ST Video Monitor IP Core

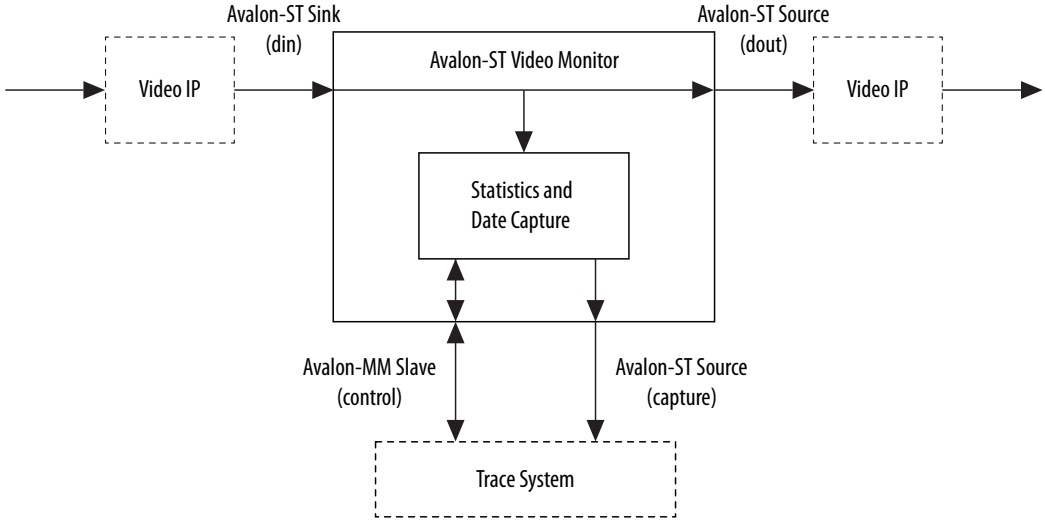
The Avalon-ST Video Monitor IP core is a debugging and monitoring component.

The Avalon-ST Video Monitor IP core together with the associated software in System Console captures and visualizes the flow of video data in a system. You can inspect the video data flow at multiple levels of abstraction from the Avalon-ST video protocol level down to raw packet data level.

The Avalon-ST Video Monitor IP core enables the visibility of the Avalon-ST video control and data packets streaming between video IP components. To monitor the video control and data packets, you must insert the monitor components into a system.

Figure 87. Avalon-ST Video Monitor Functional Block Diagram

This figure shows the monitor components in a system.



The monitored Avalon-ST video stream enters the monitor through the `din` Avalon-ST sink port and leaves the monitor through the `dout` Avalon-ST source port. The monitor does not modify, delay, or stall the video stream in any way. Inside the monitor, the stream is tapped for you to gather statistics and sample data. The statistics and sampled data are then transmitted through the capture Avalon-ST source port to the trace system component. The trace system component then transmits the received information to the host. You may connect multiple monitors to the Trace System IP core.

Note: System Console uses the `sopcinfo` file (written by Platform Designer) or the `.sof` (written by the Intel Quartus Prime software) to discover the connections between the trace system and the monitors. If you instantiate and manually connect the trace system and the monitors using HDL, System Console will not detect them.

26.1. Packet Visualization

System Console's Trace Table View contains a tabular view for displaying the information the monitors send out.

You can inspect the details of a video packet when you select a row in the trace table. The table offers the following detailed information:

- Statistics—Data flow statistics such as backpressure.
- Data—The sampled values for up to first 6 beats on the Avalon-ST data bus. `[n]` is the nth beat on the bus.
- Video control—Information about Avalon-ST video control packet.
- Video data—Packet size, the number of beats of the packet.

Note: When you turn the pixel capture feature, the packet displays a sub-sampled version of the real-time image packet in the video data section.

Table 84. Statistics

The table below lists the description of the available data flow statistics.

Statistics	Description
Data transfer cycles (beats)	The number of cycles transferring data.
Not ready and valid cycles (backpressure)	The number of cycles between start of packet and end of packet—the sink is not ready to receive data but the source has data to send.
Ready and not valid cycles (sink waiting)	The number of cycles between start of packet and end of packet—the sink is ready to receive data but the source has no data to send.
Not ready and not valid cycles	The number of cycles between start of packet and end of packet— the sink is not ready to receive data and the source has no data to send.
Inter packet valid cycles (backpressure)	The number of cycles before start of packet—the sink is not ready to receive data but the source has data to send.
Inter packet ready cycles	The number of cycles before start of packet—the sink is ready to receive data but the source has no data to send.
Backpressure	[(Not ready and valid cycles + Inter packet valid cycles) / (Data transfer cycles + Not ready and valid cycles + Ready and not valid cycles + Not ready and not valid cycles + Inter packet valid cycles)] × 100 *Note:* Inter packet ready cycles are not included in the packet duration. A packet begins when a source is ready to send data.
Utilization	[Data transfer cycles / (Data transfer cycles + Not ready and valid cycles + Ready and not valid cycles + Not ready and not valid cycles + Inter packet valid cycles)] × 100 *Note:* Inter packet ready cycles are not included in the packet duration. A packet begins when a source is ready to send data.

26.2. Monitor Settings

The capture settings panel of the trace table provides convenient access to the monitor settings.

You can change the monitor settings with the `trace_write_monitor` and `trace_read_monitor` TCL commands. At the hardware level, you can access the register map through the control Avalon-MM slave port of the monitor component.

The capture settings panel offers three options.

- **Enable**—sends of statistics and sampled data.
- **Disable**—blocks the sending of statistics and sampled data.
- **Enable with pixel capture**— the monitor starts sampling the actual pixel data in the video data packets, and displays the captured pixels in the detailed event view.

The **Capture Rate per 1000000** parameter controls the pixel percentage from randomly sampled data packets. A higher capture rate (closer to 1000000) displays a higher pixel percentage in the sample.

- If the capture rate is 5000 out of 1000000 pixels, the monitor attempts to sample one in every 200 pixels.
- If the monitor captures all the 1000000 pixels available, the monitor samples every pixel in the image.
- If there is not enough bandwidth to sample every pixel in the image, the reconstructed image may have a black and purple checkerboard pattern.

Assign a smaller capture rate value to allow the trace system to send all the debugging information through and avoid the checkerboard pattern.

26.3. Avalon-ST Video Monitor Parameter Settings

Table 85. Avalon-ST Video Monitor Parameter Settings

Parameter	Value	Description
Bits per pixel per color plane	4–20, Default = **8**	Select the number of bits per pixel (per color plane).
Number of color planes	1–3, Default = **3**	Specify the number of color planes transmitted.
Color planes transmitted in parallel	**On** or Off	• Turn on to transmit all the color planes at the same time in parallel. • Turn off to transmit all the color planes in series.
Pixels in parallel	**1**, 2, or 4	Specify the number of pixels in parallel that the video pipeline is configured for. *Note:* You must specify this parameter value to 1 to capture video data frames.
Bit width of capture interface(s)	• 8 • 16 • **32** • 64 • 128	Select the data bus width of the Avalon-ST interface sending the captured information.
Capture video pixel data	On or **Off**	Turn on to enable the inclusion of hardware that allows the monitor to capture video data frames. *Note:* This parameter only functions if you specify the number of pixels in parallel to a value of 1.

26.4. Avalon-ST Video Monitor Control Registers

Table 86. Avalon-ST Video Monitor Register Map

Address	Register	Description
0	`Identity`	Read only register—manufacturer and monitor identities. • Bits 11:0 are identities for the manufacturer, Intel = 0×6E • Bits 27:12 are identities for the monitor, Avalon-ST video monitor = 0×110
1	`Configuration Information`	For use of System Console only.
2	`Configuration Information`	For use of System Console only.
3	`Configuration Information`	For use of System Console only.
4	`Control`	• Setting bits 0 and 8 to 1 sends statistic counters. • Setting bits 0 and 9 to 1 sends up to first 6 beats on the Avalon-ST data bus. • Setting bit 0 to 0 disables both the statistics and beats.
5	`Control`	• Bits 15:0 control the linear feedback shift register (LFSR) mask for the pixel capture randomness function. The larger the mask, the less randomness is used to calculate the position of the next pixel to sample. • Bits 31:16 control the minimum gap between sampled pixels. The larger the gap, the more constant is applied to calculate the position of the next pixel.

27. VIP IP Core Software Control

The VIP Suite IP cores that permit run-time control of some aspects of their behavior use a common type of Avalon-MM slave interface.

Each slave interface provides access to a set of control registers which must be set by external hardware. You must assume that these registers power up in an undefined state. The set of available control registers and the width in binary bits of each register varies with each control interface.

The first two registers of every control interface perform the following two functions (the others vary with each control interface):

- Register 0 is the Go register. Bit zero of this register is the Go bit. A few cycles after the function comes out of reset, it writes a zero in the Go bit (remember that all registers in Avalon-MM control slaves power up in an undefined state).
- Although there are a few exceptions, most Video and Image Processing Suite IP cores stop at the beginning of an image data packet if the Go bit is set to 0. This allows you to stop the IP core and to program run-time control data before the processing of the image data begins. A few cycles after the Go bit is set by external logic connected to the control port, the IP core begins processing image data. If the Go bit is unset while data is being processed, then the IP core stops processing data again at the beginning of the next image data packet and waits until the Go bit is set by external logic.
- Register 1 is the `Status` register. Bit zero of this register is the `Status` bit; the function does not use all other bits. The function sets the `Status` bit to 1 when it is running, and zero otherwise. External logic attached to the control port must not attempt to write to the `Status` register.

The following pseudo-code illustrates the design of functions that double-buffer their control (that is, all IP cores except the Gamma Corrector and some Scaler II parameterizations):

```
go = 0;
while (true)
{
    read_non_image_data_packets();
    status = 0;
    while (go != 1)
        wait;
    read_control(); // Copies control to internal registers
    status = 1;
    send_image_data_header();
    process_frame();
}
```

For IP cores that do not double buffer their control data, the algorithm described in the previous paragraph is still largely applicable but the changes to the control register will affect the current frame.

Most VIP Suite IP cores with a slave interface read and propagate non-image data packets from the input stream until the image data header (0) of an image data packet has been received. The status bit is then set to 0 and the IP core waits until the Go bit is set to 1 if it is not already. Once the Go bit is set to 1, the IP core buffers control data, sets its status bit back to 1, and starts processing image data.

Note: There is a small amount of buffering at the input of each VIP Suite IP core and you must expect that a few samples are read and stored past the image data header even if the function is stalled.

You can use the Go and `Status` registers in combination to synchronize changes in control data to the start and end of frames. For example, suppose you want to build a system with a Gamma Corrector II IP core where the gamma look-up table is updated between each video frame.

You can build logic (or program a Nios II processor) to control the gamma corrector as follows:

1. Set the Go bit to zero. This causes the IP core to stop processing at the end of the current frame.
2. Poll the `Status` bit until the IP core sets it to zero. This occurs at the end of the current frame, after the IP core has stopped processing data.
3. Update the gamma look-up table.
4. Set the Go bit to one. This causes the IP core to start processing the next frame.
5. Poll the `Status` bit until the IP core sets it to one. This occurs when the IP core has started processing the next frame (and therefore setting the Go bit to zero causes it to stop processing at the end of the next frame).
6. Repeat steps 1 to 5 until all frames are processed.

This procedure ensures that the update is performed exactly once per frame and that the IP core is not processing data while the update is performed.

When using IP cores which double-buffer control data, a more simple process may be sufficient:

1. Set the Go bit to zero. This causes the IP core to stop if it gets to the end of a frame while the update is in progress.
2. Update the control data.
3. Set the Go bit to one.

The next time a new frame is started after the Go bit is set to one, the new control data is loaded into the IP core.

The reading on non-video packets is performed by handling any packet until one arrives with type 0. This means that when the Go bit is checked, the non-video type has been taken out of the stream but the video is retained.

27.1. HAL Device Drivers for Nios II SBT

To facilitate implementing the control mechanism and to access other registers, the VIP IP cores that permit run-time control come with a device driver.

This VIP driver is a software component that interfaces the core to the Nios II hardware abstraction layer (HAL). The VIP driver provides a high-level C++ application programming interface (API) that abstracts away the details of the register map of each VIP IP core and makes the user-control code more readable. This API is:

- Extensible—Using the C++ inheritance mechanism, it is easy to augment the API of existing cores or create classes for custom VIP IP cores.
- Optional—If the drivers are not required or you prefer to write your own light-weight implementation, simply disable the drivers with the Board Support Package (BSP) editor to remove them from the HAL to reduce the Nios II code footprint.
- Subject to minor changes in future releases—Download the VIP design examples from the Design Store for examples and refer to the documented header files that are automatically generated by the Nios II SBT in the HAL/inc directory for the full API.

28. Security Considerations

The overall system security when using a Video and Imaging Processing Intel FPGA IP is within of the control of the integrator. However, Intel recommends that you take note of certain potential security risks that may exist.

28.1. Using Avalon-ST Video Monitor Debug Tools

The AV-ST video monitors are provided for debug purposes when developing a system. The operation of these modules is to snoop the data bus they are connected to. The data they collect is passed by JTAG to the SystemConsole software in an unencrypted format. It is recommended that these modules are removed from any production releases of a design.

28.2. Configuring Memory Subsystems

A number of VIP modules require connection to RAM. Configuring these blocks will require setting a base address. The other configuration options will determine the RAM space required by the module. These address space the module operates over will be defined by these two elements. The module will only use the address space defined by these parameters. It is left to the integrator to ensure that modules are appropriately mapped in the address space of the system and they handle any protections associated with the memory subsystem that they require. The IP does not perform any checks on its configuration within a system. The Platform Designer System Integration Tool performs some limited checks on the system configuration which may or may not be sufficient according to the complexity of requirements.

29. Video and Image Processing Suite User Guide Archives

For the latest and previous versions of this document, refer to: Video and Image Processing Suite User Guide. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

30. Document Revision History for the Video and Image Processing Suite User Guide

Document Version	Intel Quartus Prime Version	Changes
2022.09.29	22.1	Added mode 1 to *Clocked Video Output Features*
2022.04.04	22.1	• Added support for Intel Agilex devices. • Added Frame Buffer II to *Minimum and Maximum Supported X and Y Resolutions* table
2021.02.12	19.4	Removed Clocked Video Input IP and Clocked Video Output IP.
2020.12.12	19.4	Updated *Clipper IP Parameters*
2020.10.30	19.4	• Enhanced H-sync and V-sync polarity description in *Clocked Video Output II Registers* table. • Added extra info to *Locked Frame Rate Conversion*.
2020.03.10	19.4	Corrected CVI to CVO in the sentence: • A CVO IP core can take in the locked PLL clock and the SOF signal.
2020.02.06	19.4	Extended the `ready` signal to high before cycle 0 on *Avalon-ST Video Packet Transmission (Symbols in Parallel)* figure.
2019.12.10	19.4	Added Warp Intel FPGA IP.
2019.07.10	19.1	Edited typo in lines 99 and 112 in `testbench/bfm_drivers.sv`. (step 1c) in the *Running the Test in Intel Quartus Prime Pro Edition* section.
2019.04.22	19.1	• Added support for Intel Stratix 10 devices. • Added a new section about interfacing the clocked video output with the SDI II IP core. The *Clocked Video and SDI II TX Interface* section provides information about the behavior of the relevant signals and the supported cadence. • Updated the 12G-SDI section to include more information about the support for 6G-SDI and 12G-SDI and added specific sections for CVI II and CVO II IP cores that also include the SDI resampler information. • Added 12G-SDI in the list of supported features for CVO II IP core in the *Supported Features for Clocked Video Output IP Cores* section. • Added a note that for 3G, 6G, and 12G modes, the CVI II IP core only supports YCbCr input color format with either 8 or 10 bits for each component.in the *Clocked Video Input Format Detection* section. • Added information about SDC warning messages for CVI II and CVO II IP cores in the *Timing Constraints* section. • Added information about the Sdi_resampler module in the *Modules for Clocked Video Input II Intel FPGA IP* section. • Updated the *Clocked Video Input II Parameter Settings* section. — Added description for the **Support 6G and 12G-SDI** parameter. Turn on this parameter to enable 6G-SDI or 12G-SDI support. — Updated the description for the **Width of vid_std_bus** parameter to include information about embedded sync. The width of the `vid_std` bus must be 3 bits.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Updated the *Clocked Video Output II Parameter Settings* section. — Added description for the **Support 6G and 12G-SDI** parameter. Turn on this parameter to enable 6G-SDI or 12G-SDI support. — Added description for the **Default SDI video standard** parameter. This parameter allows you to set the default SDI standard. To use 6G-SDI or 12G-SDI standards, you must turn on **Support 6G and 12G-SDI**. — Updated the description for the **Width of vid_std_bus** parameter to include information about embedded sync. The width of the `vid_std` bus must be 3 bits. • Added description for the `sdi_cvo_rden` signal in the *Clocked Video Output II Interface Signals* section. This signal connects to the SDI II IP core's `tx_dataout_valid` output signal. • Updated the *Behavior When Unexpected Fields are Received* section in the *Deinterlacer II IP Core* chapter. • Added a new option (**512**) for the **Avalon-MM master(s) local ports width** parameter in the *Deinterlacer II Parameter Settings* section. • Removed outdated information and updated the description in the *Deinterlacing Control Registers* section. • Added a note that for security reasons, you should remove the Trace System components from designs going to production in the *Trace System IP Core* section. This IP core is meant only for Intel Quartus Prime Standard Edition environments. • Removed the information about `avi2raw` and `raw2avi` utilities from the *Avalon-ST Video Verification IP Suite* appendix section because this functionality is available in the open source FFmpeg tool (ffmpeg.org).
2018.09.24	18.1	• Edited the *Device Family Support* section to clearly indicate that the VIP Suite IP cores support Arria V GZ and Cyclone IV E variants. • Added new features for the Deinterlacer II IP core: — New run-time control and status registers to improve deinterlacing quality for mid-range motion-adaptive configurations. — New run-time control registers to allow run-time switching between "bob", "weave" and "motion adaptive" modes. • Added Native 4K HDR/SDR Passthrough support information for the Deinterlacer II IP core in the *Deinterlacing Algorithm Options* section. • Updated the 4K Video Passthrough Support section to add information about determining best approaches for 4K passthrough for Deinterlacer II IP core configurations. • Edited the existing and added new registers for the Deinterlacer II IP core in the *Deinterlacing Control Registers* section. • Updated the description for the parameters for the Deinterlacer II IP core: • Updated the introduction description for the Test Pattern Generator II IP core. The IP core now offers five test pattern options and supports up to 8 pixels per cycle. • Added information about the five test patterns for the Test Pattern Generator II IP core: Color bars, Grayscale bars, Black and White bars, SDI Pathological, and Uniform Background • Added information about output subsampling and color space for the Test Pattern Generator II IP core.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Updated or added the following parameters for the Test Pattern Generator II IP core: — **Output format** — **Default interlacing** — **Run-time control** — **Reduced control register readback** — **Pipeline dout ready** — **Enable border for bar patterns** — **Uniform** — **Number of test patterns** — **Subsampling & Colorspace** • Updated the *Test Pattern Generator II Control Register Map* table with the latest addresses and registers. • Updated the *Mixer II IP Core* introduction section to add information about the new **Synchronize background to layer 0** parameter. • Updated the Mixer II feature information to add that now the Mixer II IP core supports up to 20 inputs and up to 8 pixels in parallel.. • Updated or added following parameters for the Mixer II IP core: — **Number of inputs** — **Alpha Blending Enable** — **Input*N* alpha channel** — **Synchronize background to layer 0** — **Reduced control register readback** — **Add extra register stages to data pipeline** • Updated the *Alpha Blending Enable* section for the Mixer II IP core to add information about the new **Input*N* alpha channel** parameter. • Updated the description for the `Background Width` and `Background Height` Mixer II registers to add that these registers are not available when the **Synchronize background to layer 0** parameter is enabled. • Add a new section on *Low-Latency Mode* for the Mixer II IP core. • Updated the Switch II feature information to add that now the Switch II IP core supports up to 8 pixels in parallel and added a note that the function of the status bit for the Switch II IP core differs from the status bits of the other VIP IP cores.
2018.07.20	18.0	Corrected the error in the description for the Scaler II input and output frame widths and heights in the Scaler II parameter settings. The supported maximum widths and heights are 8,192.
2018.05.07	18.0	• Added a table in the *Video and Image Processing IP Cores* introduction section to list the minimum and maximum input/output resolutions. • Edited the Progressive interlacing values in the *Avalon-ST Video Control Packet Interlaced Nibble Decoding* table. • Added a note in the *Avalon-ST Operation* section that some VIP IP cores require a control packet to initialize storage. To ensure correct operation across all VIP components, at least one control packet must be sent to an IP core prior to any video packets. • Added a caution note in the *Avalon-ST Video Support* section that some memory configurations for motion adaptive configurations of the Deinterlacer II (but not video over film configurations) can result in motion information being incorrectly overwritten by information from subsequent lines, resulting in a reduction in QOR manifesting in weave artifacts.

Document Version	Intel Quartus Prime Version	Changes
		• Added a note that the VIP IP cores support only ModelSim simulation software in the *Specifying IP Core Parameters and Options* section. • Edited the register bits width for the Frame Buffer II IP core. The actual width is 32 bits. • Updated the maximum frame width and height from 4096 to 8192 for the following FPGA IP cores: — Avalon-ST Video Stream Cleaner — Clipper II — Clocked Video Input II — Deinterlacer II — Frame Buffer II — Mixer II — Scaler II — Test Pattern Generator II • Added a new option (**8**) for the **Number of pixels in parallel** parameter for the following IP cores: — 2D FIR Filter II — Avalon-ST Video Stream Cleaner — Chroma Resampler II — Clipper II — Clocked Video Input II — Clocked Video Output II — Color Space Converter II — Deinterlacer II — Frame Buffer II — Gamma Corrector II — Configurable Guard Bands — Interlacer II — Mixer II — Scaler II — Switch II — Test Pattern Generator II • Added information about the `acknowledge` bit for the `Misc` register in the *Frame Buffer Control Registers* section. • Removed note that stated the Clocked Video Output II IP core does not support Genlock. The Clocked Video Output II IP core now supports Genlock. • Added new parameters for Clocked Video Output II IP core: — **Generate Synchronization outputs** — **Accept Synchronization inputs** — **Set vco_clk_divider increment to pixels in parallel** • Added Genlock signals for Clocked Video Output II IP core: — `vid_vcoclk_div` — `vid_sof` — `vid_sof_locked` — `sof` — `sof_locked` • Added information that `ModeN SOF Line` is a 13-bit register and `ModeN Vcoclk Divider` is a 14-bit register in the *Clocked Video Output II Registers* table. • Edited the description for the `sof_locked` signal in the *Clocked Video Output Signals* table. This signal is an input signal.

Date	Version	Changes
November 2017	2017.11.06	• Updated support for Intel Cyclone 10 LP and Intel Cyclone 10 GX devices as final. • Changed the term Qsys to Platform Designer. • Made extensive changes to the *Avalon-ST Video Verification IP Suite* section based on the changes in the Intel Quartus Prime software. • Changed the minimum width and height values to 32 in the *Clipper II Parameter Settings* section. • Added a note that if you have enabled run-time control, the Go bit gets deasserted by default for the following IP cores: — 2D FIR Filter II — Clipper II — Deinterlacer II — Scaler II — Frame Buffer II — Configurable Guard Bands — Interlacer II — Scaler II — Test Pattern Generator II • Added a note that the following IP cores require even frame widths when using 4:2:2 data; odd frames widths create unpredictable results or distorted images: — 2D FIR Filter II — Deinterlacer II — Scaler II — Configurable Guard Bands — Scaler II — Mixer II — Test Pattern Generator II • Added information that if new frame dimensions are set, the Frame Buffer II IP core requires a write to the `Frame Start Address` register for the new settings to take effect. • Added new parameters for Frame Buffer II IP core: — **Delay length (frames)** — **Drop/repeat user packets** • Edited the **Reduced control register readback** parameter information for the Gamma Corrector IP core that only values written to registers 5 and 6 in the control slave interface can be read back. Contents of registers 6 and above cannot be read back in any mode. • Added a note in the *Gamma Corrector Control Registers* section that the values written to registers 6 and above cannot be read back in any mode. • Updated the *Memory Map for Frame Reader* section with information about ancillary packets and edited the *Memory Map for Base Address 0x1000_0000 for Non 8-Bit Pixel Values* figure with the correct mapping. • Updated the *Clocked Video Output II Control Registers* section. Some minor changes were made to the common control registers to correct reported bugs and limitations. • Added a new register (`Motion scale`) to the *Deinterlacer II Control Registers* section. • Added information and changed the title of the *Tuning Motion Shift* section to .
May 2017	2017.05.10	Republished to add some missing chapters due to technical glitch.

continued...

Date	Version	Changes
May 2017	2017.05.08	• Rebranded as Intel. • Added preliminary device support for Intel Cyclone 10 LP and Intel Cyclone 10 GX devices. • Removed all information about the following IP cores. These IP cores are no longer supported in Intel Quartus Prime versions 17.0 and later. — 2D FIR Filter — Alpha Blending Mixer — Chroma Resampler — Color Space Converter — Color Plane Sequencer — Deinterlacer — Frame Buffer — Frame Reader — Gamma Corrector — Interlacer — Switch • Added information for the Configurable Guard Bands IP core. This is a new IP core being released in version 17.0. • Updated the performance and resource data information based on an Arria 10 device. • Updated the stall behavior and error recovery information for 2D FIR Filter II, CVO/CVO II, Color Plane Sequencer II, and Deinterlacer II IP cores. • Added or updated these chapters: — *Clocked Video* — *VIP Run-Time Control* — *VIP Connectivity Interfacing* — *VIP Software Control* • Reorganized the signals, parameters, and registers information according to the respective Clocked Video Interface IP cores. • Updated the description for the `vid_color_encoding` and `vid_bit_width` signals in the CVI II IP core. Tie these signals to low if no equivalent signals are available from the IP core driving the CVI II. • Updated GUI information for the Chroma Resampler II IP core. Added the following parameters: — **Enable vertical luma adaptive resampling** — **Vertical chroma siting** — **Variable 3 color interface** — **Enable 4:2:0 input** — **Enable 4:2:0 output** • Updated GUI information for the Deinterlacer II IP core. — Removed the **How user packets are handled: Pass all user packets through to the output** and **Enable embedded chroma resamplers** parameters. — Edited the description for the **Deinterlacing algorithm**, **Fields buffered prior to output**, and **Use separate clock for the Avalon-MM master interface(s)** parameters. • Changed `Cadence Detect On` register to `Cadence Detect and advanced tuning registers On` register for the Deinterlacer II IP core. This updated register enables the cadence detection feature and (if configured) the video over film feature together with all the motion and cadence/VOF tuning registers. • Updated the Scaler II calculation for the nearest neighbor algorithm. • Removed edge sharpening feature for the Scaler II IP core. • Clarified the GUI information for the Test Pattern Generator II IP core.

continued...

Date	Version	Changes
October 2016	2016.10.31	• Added information about these new IP cores: — 2D FIR Filter II — Chroma Resampler II — Color Plane Sequencer II — Gamma Corrector II — Interlacer II • Added a flowchart to illustrate the behavior of the VIP IP cores. • Updated information for Deinterlacer II IP core (this IP core is now merged with the Broadcast Deinterlacer IP core). — Updated the Deinterlacer II parameter settings table to include the new parameters. — Added new information about Avalon-ST Video and 4K Video passthrough support. — Updated the Motion Adaptive mode bandwidth requirements to reflect the upgraded Deinterlacer II IP core. • Updated information for Clocked Video Output IP cores. — Updated mode bank selection information for CVO and CVO II IP cores. You can configure the IP cores to support 1 to 13 modes. — Added information to enable the `Go` bit for both CVO IP cores to avoid situations where a write in the streaming side cannot be issued to the video clock side because the video clock isn't running. — Added new parameter for CVO II IP core: **Low latency mode**. Setting this parameter to 1 enables the IP core to start timing for a new frame immediately • Updated information for Clocked Video Input II IP core. — Added three new parameters: **Enable matching data packet to control by clipping**, **Enable matching data packet to control by padding**, **Overflow handling**. — Added two new signals: `Clipping` and `Padding`. — Updated description for these signals: `vid_color_encoding` and `vid_bit_width`. — Updated information about the `Status` register. The register now includes bits to support clipping and padding features. • Updated information for Mixer II IP core. — Updated alpha stream information for Mixer II IP core. When you enable alpha stream, the LSB is in Alpha value and the control packets are composed of all symbols including Alpha. — Corrected the description for the `Control` and `Status` registers.
May 2016	2016.05.02	• Added information about a new IP core: Avalon-ST Video Stream Cleaner. • Frame Buffer II IP core: — Added new Frame Buffer II IP core parameters: • **Enable use of fixed inter-buffer offset** • **Inter-buffer offset** • **Module is Frame Reader only** • **Module is Frame Writer only** — Updated the default values for these Frame Buffer II IP core parameters: • Maximum frame width = 1920 • Maximum frame height = 1080 — Updated the existing and added new Frame Buffer II IP core registers. — Added new information for Frame writer-only and Frame reader-only modes.

continued...

Date	Version	Changes
		• Broadcast Deinterlacer IP core: — Updated the existing and added new Broadcast Deinterlacer IP core registers. — Edited the *Design Guidelines for Broadcast Deinterlacer IP Core* section and removed the *Active Video Threshold Adjustment* section. The information is no longer relevant. • Clocked Video Interface IP core: — Added new or updated these Clocked Video Input II IP core signals: • `dout_empty` • `vid_locked` (updated) • `vid_datavalid` (updated) • `vid_color_encoding` • `vid_bit_width` • `vid_total_sample_count` • `vid_total_line_count` — Added a new register, `Color Pattern`, for the Clocked Video Input II IP core. — Updated information for the `Standard` format for the Clocked Video Input II IP core. — Added new information for output video modes in the Clocked Video Output II IP core. — Added information that multiple pixels in parallel are only supported for external sync mode in the Clocked Video Output II IP core. — Removed the **Accept synchronization outputs** parameter and the related signals from the Clocked Video Output II IP core. • `vcoclk_div` • `sof` • `sof_locked` • `vid_sof` • `vid_sof_locked` • Mixer II IP core: — Added new or updated the following Mixer II IP core parameters: • **Number of inputs** • **Alpha Blending Enable** • **Layer Position Enable** • **Register Avalon-ST ready signals** • **Uniform values** • **Number of pixels transmitted in 1 clock cycle** • **Alpha Input Stream Enable** • **4:2:2 support** • **How user packets are handled** — Removed these Mixer II IP core parameters: • **Number of color planes** • **Run-time control** • **Output format** — Updated the existing and added new Mixer II IP core registers. — Added alpha blending information for Mixer II IP core in the *Alpha Blending - Mixer II* section. — Added information about defining layer mapping in the *Layer Mapping-Mixer II* section. • Switch II IP core: — Updated the features information to include that each input drives multiple outputs and each output is driven by one input. — Added a new register: `Din Consume Mode Enable` • Added links to archived versions of the *Video and Image Processing Suite User Guide*.

continued...

Date	Version	Changes
November 2015	2015.11.02	• Removed information about the Clipper and Test Pattern Generator IP cores. These cores are no longer supported in versions 15.1 and later. • Changed instances of *Quartus II* to *Intel Quartus Prime*. • Edited the description of the `vid_de` signal for the Clocked Input II IP core—this signal is driven by the IP core to indicate the data lines are carrying active picture. • Added two new Mixer II IP core registers. • Added conditions for the Video Mixing IP cores; if these conditions are not met, then the Mixer behavior is undefined and the core is likely to lock up. • Edited the description of the `Coeff-commit` control register for the Color Space Converter II IP core. Writing a 1 to this location commits the writing of coefficient data.
May 2015	2015.05.04	• Edited the description of the `Input (0-3) Enable` registers for the Mixer II IP core. The 1-bit registers are changed to 2-bit registers: — Set to bit 0 of the registers to display input 0. — Set to bit 1 of the registers to enable consume mode. • Edited the description of the `Interrupt` register to unused for the Color Space Converter II, Frame Buffer II (writer), and Test Pattern Generator II IP cores. • Edited the register information for the Switch II IP core: — Changed the description of the `Interrupt` register to state that bit 0 is the interrupt status bit. — Updated the description of the `Control` register to add that bit 1 of the register is the interrupt enable bit. — Edited the typo in address 15 of the Switch II IP core— Dout12 Output Control changed to Dout11 Output Control. • Edited the typos in the descriptions for `Output Width` and `Output Height` registers for the Test Pattern Generator IP cores. • Edited the parameter settings information for the Mixer II IP core. — Added description for new parameter **Pattern** which enables you to select the pattern for the background layer. — Removed information about **Color planes transmitted in parallel** . This feature is now default and internally handled through the hardware TCL file. • Edited the parameter settings information for the Frame Buffer II IP core. — Added descriptions for parameters that were not supported in the previous version: **Maximum ancillary packets per frame**, **Interlace support**, **Locked rate support**, **Run-time writer control**, and**Run-time reader control** — Removed information about **Ready latency** and **Delay length (frames)**. These features are fixed to 1 and internally handled through the hardware TCL file. • Edited the parameter settings information for the Avalon-ST Video Monitor IP core. — Added description for new parameters: **Color planes transmitted in parallel** and **Pixels in parallel**. — Removed information about the **Number of color planes in sequence** parameter. You can specify whether to transmit the planes in parallel or in series using the **Color planes transmitted in parallel** parameter. — Added a note that the **Capture video pixel data** feature only functions if you specify the number of pixels transmitted in parallel to 1.

continued...

Date	Version	Changes
January 2015	2015.01.23	• Added support for Arria 10 and MAX 10 devices. Arria 10 devices support only the following IP cores: Avalon-ST Video Monitor, Broadcast Deinterlacer, Clipper II, Clocked Video Input, Clocked Video Input II, Clocked Video Output, Clocked Video Output II, Color Space Converter II, Deinterlacer II, Frame Buffer II, Mixer II, Scaler II, Switch II, and Test Pattern Generator II. • Removed the **Generate Display Port output** parameter from the Clocked Video Output II IP core. This feature is now default and internally handled through the hardware TCL file. • Added description for a new signal for Clocked Video Input II IP core: `vid_hdmi_duplication[3:0]`. • Added information for the missed out `Coeff-commit` control register for the Color Space Converter II IP core. • Edited the description for the Frame Buffer II parameters.
August 2014	14.0	• Added new IP cores: Clocked Video Output II, Clocked Video Input II, Color Space Converter II, Mixer II, Frame Buffer II, Switch II, and Test Pattern Generator II. • Revised the performance and resource data for different configurations using Arria V and Cyclone V devices. • Added information about IP catalog and removed information about MegaWizard Plug-In Manager. • Updated bit 5 of the `Status` register as unused for the Clocked Video Input IP core. • Corrected the formula for adjusting the filter function's phase for the Scaler II IP core. • Consolidated the latency information for all IP cores in the Overview chapter. • Consolidated the stall behavior and error recovery information for all IP cores in the Overview chapter. • Moved the 'Video Formats' section from Clocked Video Input and Output chapters to the Interfaces chapter.
February 2014	13.1	• Added information on 4:2:2 support. • Added Design Guidelines section for the Broadcast Deinterlacer IP core. • Removed information about Arria GX, Cyclone, Cyclone II, Stratix, Stratix GX, Stratix II, Stratix II GX, and all HardCopy devices. Altera no longer supports these devices.
July 2013	13.0	• Added new IP cores: Broadcast Deinterlacer and Clipper II • Removed Scaler IP core. This core is no longer supported in version 13.0 and later. • Added information about the **Add data enable signal** parameter and the `vid_de` signal for Clocked Video Input IP core.
April 2013	12.1.1	Added the following information for the Avalon-ST Video Monitor IP core. • Added description for packet visualization. • Added explanation for **Capture Rate per 1000000** option for monitor settings. • Added **Capture video pixel data** parameter. • Added Control Bits entry to the register map.

continued...

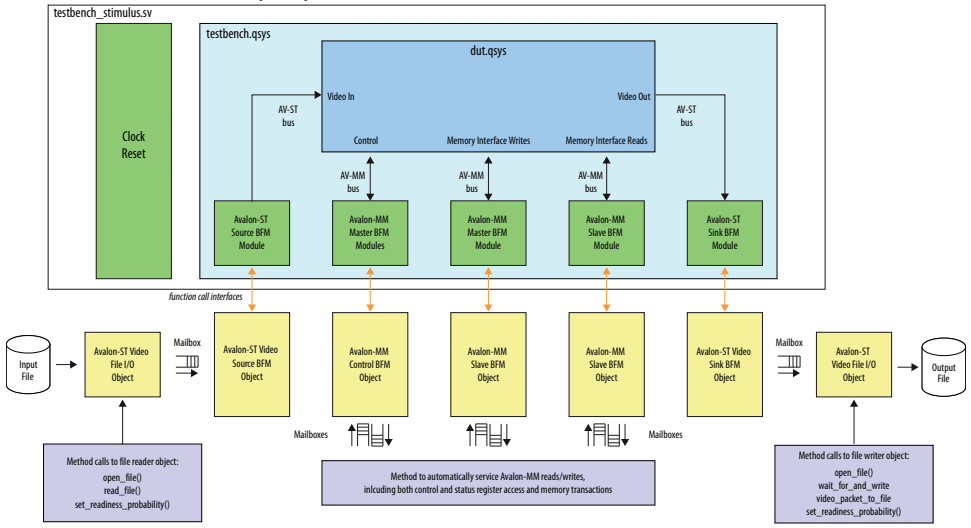
Date	Version	Changes
January 2013	12.1	• Added Deinterlacer II Sobel-Based HQ Mode information for the Deinterlacer II IP core. • Updated Table 1–17 to include latest Deinterlacer II IP core performance figures for Cyclone IV and Stratix V devices. • Edited the description of the `rst` signal for the Clocked Video Output IP core. • Added a note to explain that addresses 4, 5, and 6 in the Frame Buffer control register map are optional and visible only when the GUI option is checked. • Updated Table 23–4 to include the functionality of address 0 in the register map.
July 2012	12.0	• Added new IP cores: Avalon-ST Video Monitor and Trace System. • Added information on the edge-adaptive scaling algorithm feature for the Scaler II IP core.
February 2012	11.1	• Reorganized the user guide. • Added new appendixes: "Avalon-ST Video Verification IP Suite" and "Choosing the Correct Deinterlacer". • Updated Table 1-1 and Table 1-3.
May 2011	11.0	• Added new IP core: Deinterlacer II. • Added new polyphase calculation method for Scaler II IP core. • Final support for Arria II GX, Arria II GZ, and Stratix V devices.
January 2011	10.1	• Added new IP core: Scaler II. • Updated the performance figures for Cyclone IV GX and Stratix V devices.
July 2010	10.0	• Preliminary support for Stratix V devices. • Added new IP core: Interlacer. • Updated Clocked Video Output and Clocked Video Input IP cores to insert and extract ancillary packets.
November 2009	9.1	• Added new IP cores: Frame Reader, Control Synchronizer, and Switch. • The Frame Buffer IP core supports controlled frame dropping or repeating to keep the input and output frame rates locked together. The IP core also supports buffering of interlaced video streams. • The Clipper, Frame Buffer, and Color Plane Sequencer IP cores now support four channels in parallel. • The Deinterlacer IP core supports a new 4:2:2 motion-adaptive mode and an option to align read/write bursts on burst boundaries. • The Line Buffer Compiler IP core has been obsoleted. • The Interfaces chapter has been re-written.
March 2009	8.0	• The Deinterlacer IP core supports controlled frame dropping or repeating to keep the input and output frame rates locked together. • The Test Pattern Generator IP core can generate a user-specified constant color that can be used as a uniform background. • Preliminary support for Arria II GX devices.

A. Avalon-ST Video Verification IP Suite

The Avalon-ST Video Verification IP Suite provides a set of SystemVerilog classes (the class library) that you can use to ensure that a video IP simulates correctly and conforms to the Avalon-ST video standard.

Figure 88. Test Environment for the Avalon-ST Video Class Library

The figure below shows the elements in the Avalon-ST Video Verification IP Suite. Yellow indicates the class library components of the test environment, green indicates the Avalon-ST Bus Functional Model (BFM) as instanced in the Platform Designer environment, purple indicates the test method calls themselves, and blue indicates the device under test (DUT).



The DUT is fed with Avalon-ST Video-compliant video packets and control packets. The packets are either constrained randomly or derived from a test video file. The responses from the DUT are collected, analyzed, and any resultant video written to an output file.

The class library uses the Avalon-MM and Avalon-ST source and sink BFMs [1] and provides the following functionality:

- Embodies the pixels-in-parallel upgrades to the Avalon-ST Video standard to facilitate compliance testing.
- Implements a host of common Avalon-ST Video protocol failures that the DUT can be tested against. You can configure these using simple method calls to the class library.
- Implements file reader or file writer functionality to facilitate DUT testing with real video sequences.
- Uses SystemVerilog's powerful verification features such as mailboxes and randomization of objects. These features allow you to easily construct complex and noisy bus environments for rigorous stress-testing of DUTs.

A.1. Avalon-ST Video Class Library

The class library is a unified modeling language (UML)-styled class structure broken down into individual files and packages.

Figure 89. UML-Style Class Diagram

The figure shows a unified modeling language (UML)-styled diagram of the class structure of the library and how these break down into individual files and packages.

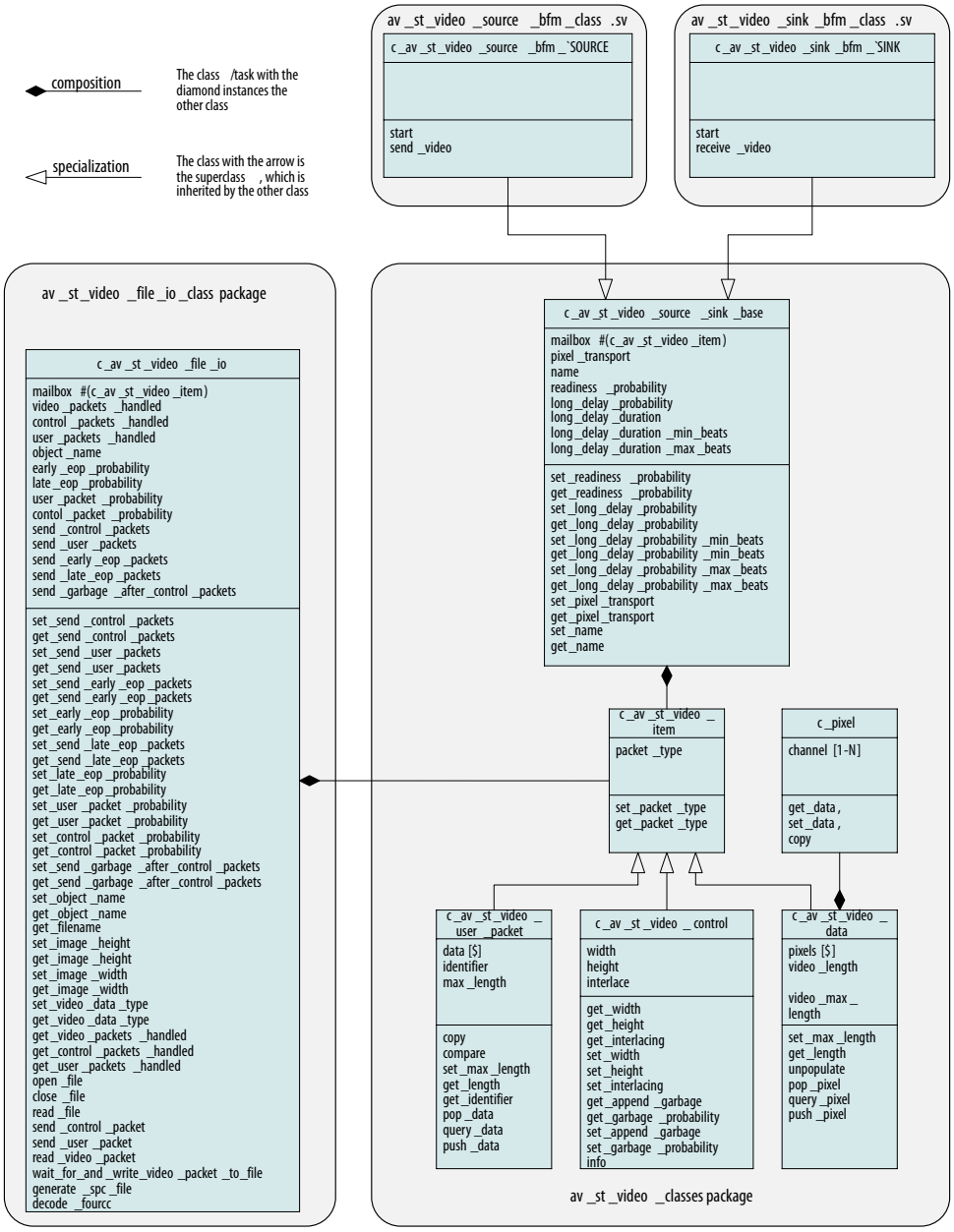


Table 87. Class Description

The table describes each of the classes in the *av_st_video_classes* package.

Note: The classes listed do not contain information about the physical transport mechanism and the Avalon-ST Video protocol.

Class	Description
class c_av_st_video_item	The most fundamental of all the classes. Represents any item that is sent over the Avalon-ST bus and contains a `packet_type` field. You can set the field to `video_packet`, `control_packet`, or `user_packet` types. These three packet types are represented by classes which extend this base class. Structuring the classes in this way allows you to define the mailboxes and queues of *c_av_st_video_item*. Then, you can send any type of packet in the order that they appear on the bus.
class c_pixel	Fundamental and parameterized class. Comprised of an array of channels that contains pixel data. For example, a pixel from an RGB24 video system comprises an array of three channels (8 bits per channel). A pixel for a YcbCr system comprises two channels. An individual channel either represents a luminance or chroma-type component of video data, one RGB component, or one alpha component. The class provides "getters", "setters", and "copy" methods. The parameters for this class are **BITS_PER_CHANNEL** and **CHANNELS_PER_PIXEL**.
class c_av_st_video_data	Parameterized class. Contains a *queue* of pixel elements. This class library is used by other classes to represent fields of video and line (or smaller) units of video. It extends *c_av_video_item*. The class provides methods to push and pop pixels on and off the queue. The parameters for this class are **BITS_PER_CHANNEL** and **CHANNELS_PER_PIXEL**.
class c_av_st_video_control	Parameterized class. Extends *c_av_video_item*. Comprises of width, height, and interlaced bits (the fields found in an Avalon-ST video control packet). It also contains data types and methods that control the addition of garbage beats that are used by other classes. The class provides methods to get and set the individual fields. The parameters for this class are **BITS_PER_CHANNEL** and **CHANNELS_PER_PIXEL**.
class c_av_st_user_packet	Parameterized class. Contains a *queue* of data and is used by the other classes to represent packets of user data. It extends *c_av_video_item*. The class provides methods to push and pop data on and off the queue. The parameters for this class are **BITS_PER_CHANNEL** and **CHANNELS_PER_PIXEL**.

Table 88. Additional Class Description

The table describes the classes included in the *av_st_video_file_io_class* package, and the source and sink class packages.

Class	Description
class c_av_st_video_source_sink_base	Designed to be extended by source and sink BFM classes. Contains a mailbox of *c_av_st_video_item*, together with various fields that define the transport mechanism (serial or parallel), record the numbers of packets sent, and define the service quality (readiness) of the source or sink.
class c_av_st_video_source_bfm_'SOURCE	Extends *c_av_st_video_source_sink_base*. Named according to the instance names of the Avalon-ST source and sink BFMs in the SystemVerilog netlist. This is because you must access the API functions in the Avalon-ST BFMs by directly calling them through the design hierarchy. Therefore, this

continued...

Class	Description
	hierarchy information is required in the Avalon-ST video source and sink classes. This means that a unique class with the correct design hierarchy information for target source or sink is required for every object created of that class type. To overcome this limitation, create the source and sink class files (*av_st_video_bfm_class.sv* and *av_st_video_sink_bfm_class.sv*) which are designed to be `included into the test environment with `defines set to point to the correct hierarchy. The source class comprises of a simple `start()` task and a `send_video` task (called by the start task). The `send_video` task continually polls its mailbox. When a `video_item` arrives, the `video_item` is assembled into a set of transactions according to its type and the transport mechanism specified. Then, the `video_item` is sent to the Avalon-ST BFM. One Avalon-ST BFM transaction is considered as one beat on the Avalon-ST bus, comprised of the logic levels on the SOP, EOP, READY, VALID, and EMPTY signals, as well as the data on the bus in a given clock cycle. For example, a video packet is sent to the BFM preceded by a 0x0 on the LSB of the first transaction, as per the Avalon-ST video protocol. A control packet is preceded by a 0xf on the LSB. Then, the height, width and interlacing fields are sent in subsequent transaction in accordance to the Avalon-ST Video protocol. The class *c_av_st_video_source_bfm_`SOURCE* requires you to create an object from it and to call the `start()` task as it automatically handles any `video_item` sent to its mailbox. No other interaction is required.
class c_av_st_video_sink_bfm_'SINK	Operates in the same way as the source class, except it contains a `receive_video()` task and performs the opposite function to the source. This class receives incoming transactions from the Avalon-ST sink BFM, decoding their type, assembling them into the relevant objects (control, video, or user packets), and pushing them out of its mailbox. No further interaction is required from the user.
class c_av_st_video_file_io	Parameterized class. Extends *c_av_video_item*. Comprised of width, height, and interlaced bits (the fields found in an Avalon-ST video control packet). It also contains data types and methods that control the addition of garbage beats that are used by other classes. The class provides methods to get and set the individual fields.
class c_av_st_user_packet	This parameterized class is defined in a separate file (*av_st_video_file_io_class.sv*) because some test environments do not use video data from a file, using constrained random data generated by the other classes instead. This class provides the following methods: • to read and write video files (in `.raw` format) • to send or receive videos and control packet objects to or from the mailbox. Variables that govern the file I/O details include the ability to artificially lengthen and shorten video packets and to introduce garbage beats into control packets by various get and set method calls. Typical usage of the file I/O class is to construct two objects—a reader and a writer, call the open file methods for both, call the `read_file` method for the reader, and repeatedly call the `wait_for_and_write_video_packet_to_file` method in the writer. The parameters for this class are **BITS_PER_CHANNEL** and **CHANNELS_PER_PIXEL**.

A.2. Example Tests

An example system is available in the Intel Quartus Prime install directory.

To try out some of the class library, run the example tests on the example DUT by following the steps given.

Note: The actual commands used in this section are for a Linux example. However, a similar flow applies to Windows.

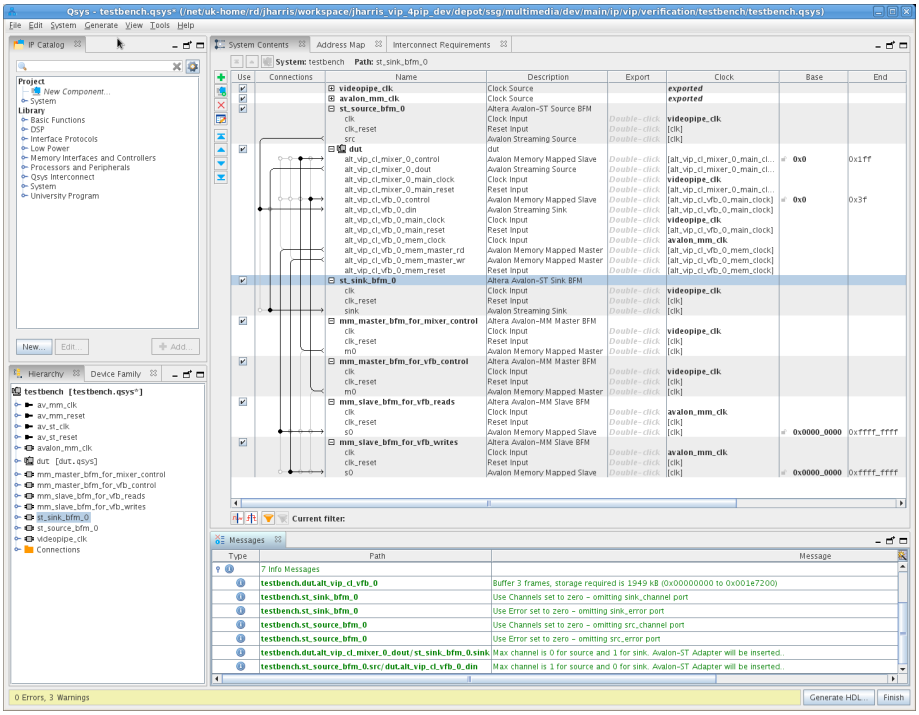
A.2.1. Generating the Testbench Netlist

1. Copy the verification files from `$(QUARTUS_ROOTDIR)/../ip/altera/vip/verification` to a local directory.
2. Change the directory to where you copied the files to and ensure that write permissions exist on `testbench/testbench.qsys` and `dut/dut.qsys` so that the system can be saved prior to generation.
3. Create an ipx file pointing to the DUT: `>ip-make-ipx --source-directory=dut/`
4. **Skip this step if you are using Intel Quartus Prime Standard Edition**. If you use Intel Quartus Prime Pro Edition, create a new project in the Intel Quartus Prime software before you generate the testbench.
   a. Next, start the Platform Designer system integration tool (Qsys).
   b. To select the Platform Designer system, browse to `testbench` and select `testbench.qsys`.
   c. Click **Open** and **OK** to convert the project to the Intel Quartus Prime Pro Edition format.
5. **Skip this step if you are using Intel Quartus Prime Pro Edition**. Start the Platform Designer system integration tool from the Intel Quartus Prime software (**Tools ➤ Platform Designer** or through command line.

```
>cd testbench
>qsys-edit testbench.qsys
```

6. The system refreshes and shows an example DUT. In this instance, the example DUT is another Platform Designer system comprised of the Mixer II and Frame Buffer II IP cores. You can easily replace this example by any other VIP IP cores or user IP functions. None of the interfaces are exported. All of the DUT Avalon-MM and Avalon-ST I/Os are attached to the BFM, which in turn interfaces to the class library.

Figure 90. Platform Designer Dialog Box



7. Create the `testbench.v` netlist from the Platform Designer project by clicking **Generate HDL**, set **Create simulation model** to **Verilog**. Click **Generate**. Close the **Generate completed** dialog box, and exit the Platform Designer and Intel Quartus Prime software (if open).
   Platform Designer generates the `testbench.v` netlist and all the required simulation files.

   Note: Platform Designer in the Intel Quartus Prime Pro Edition software may report that some of the IP cores have validation errors. You can safely ignore this error.

A.2.2. Running the Test in Intel Quartus Prime Standard Edition

The `class_library` folder and example tests are designed for the QuestaSim* simulator. You can also run the tests using the ModelSim - Intel FPGA Edition simulator. If the scripts detect that this simulator is being used, the cut-down `class_library_ae` folder will be substituted. You will observe some errors, but the tests will still compile and run to completion.

Run the test by changing to the example video files test or example constrained random test directory and start the simulator. To run the example video files test, type:

```
>cd $ALTERA_VIDEO_VERIFICATION/example_video_files
>vsim -do run.tcl
```

The test runs and completes with the following message:

```
"Simulation complete. To view resultant video, now run the windows raw2avi
application."
```

A.2.3. Running the Test in Intel Quartus Prime Pro Edition

Intel Quartus Prime Pro Edition requires different hierarchical paths to run.

1. When Platform Designer has generated the testbench, make the following edits:
   a. Edit line 27 in `testbench/run.tcl`.

```
27 set QSYS_SIMDIR ../testbench/testbench/sim
```

   b. Edit lines 28 and 29 in `testbench/defines.sv`.

```
28 `define MM_SINK_WR
testbench.mm_slave_bfm_for_vfb_writes.mm_slave_bfm_for_vfb_writes
```

```
29 `define MM_SINK_RD
testbench.mm_slave_bfm_for_vfb_reads.mm_slave_bfm_for_vfb_reads
```

   c. Edit lines 20, 37, 59, 74, 99, and 112 in `testbench/bfm_drivers.sv`.

```
20  `define SLAVE_HIERARCHICAL_LOCATION
testbench.mm_slave_bfm_for_vfb_reads.mm_slave_bfm_for_vfb_reads
```

```
37  `define SLAVE_HIERARCHICAL_LOCATION
testbench.mm_slave_bfm_for_vfb_writes.mm_slave_bfm_for_vfb_writes
```

```
59  `define MASTER_HIERARCHY_NAME
testbench.mm_master_bfm_for_mixer_control.mm_master_bfm_for_mixer_control
```

```
74  `define MASTER_HIERARCHY_NAME
testbench.mm_master_bfm_for_vfb_control.mm_master_bfm_for_vfb_control
```

```
99  `define SOURCE_HIERARCHY_NAME `TESTBENCH.`SOURCE.`SOURCE
```

```
112 `define SINK_HIERARCHY_NAME `TESTBENCH.`SINK.`SINK
```

2. Run the test by changing to the example video files test or example constrained random test directory and start the simulator. To run the example video files test, type:

```
>cd $ALTERA_VIDEO_VERIFICATION/example_video_files
>vsim –do run.tcl
```

The test runs and completes with the following message:

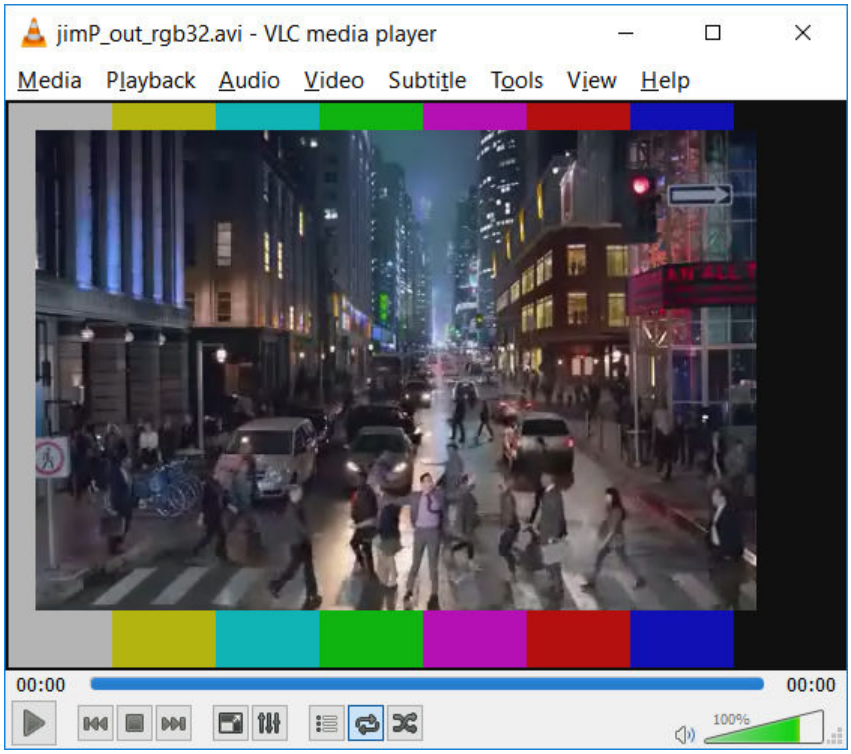
```
"Simulation complete. To view resultant video, now run the windows raw2avi
application."
```

A.2.4. Viewing the Video File

The example video files test produces a raw output video file (`jimP_out_rgb32.raw`).

You may use the raw file to create an `.avi` file for viewing through a utility such as fmpeg, such as the image below.

Figure 91. jimP_out_rgb32.avi File



View the `.avi` file with a media player. The media player shows the output from the Mixer, which is the test pattern background initially, while the frame buffer receives its first frame. Then, the test video is mixed in but offset by 20 pixels

Note: To run with other simulators, make the appropriate edits to the `verification/testbench/run.tcl` file for the simulator required, For example, `verification/testbench/testbench/simulation/cadence/ncsim_setup.sh`.

A.2.5. Verification Files

You can use the verification files used for this example as templates for your designs.

Table 89. Verification File Folders

Folder	File	Description
`class_library`	• `av_mm_class.sv` • `av_mm_control_bfm_class.sv` • `av_mm_control_classes.sv` • `av_mm_master_bfm_class_inc.sv`	The class library files contain all the systemVerilog classes plus tasks to service Avalon-MM memory requests and to read and write video to file.

continued...

Folder	File	Description
	• av_mm_slave_bfm_class_inc.sv • av_mm_split_rw_bfm_driver_fnc.sv • av_st_control_bfm_class.sv • av_st_video_classes.sv • av_st_video_file_io_class.sv • av_st_video_sink_bfm_class.sv • av_st_video_source_bfm_class.sv • tasks.sv • warning_banners.sv	
dut	• dut.qsys	The DUT in the examples is a Platform Designer system which instances the Mixer II and Frame Buffer II IP cores.
example_constrained_random	• run.tcl • test.sv	Both the constrained random and video file examples run in the same way, using a simple run.tcl script and test.sv file.
example_video_files	• jimP_rgb32.raw • jimP_rgb32.spc	Test input video.
	• run.tcl • test.sv	Both the constrained random and video file examples run in the same way, using a simple run.tcl script and test.sv file.
testbench	bfm_drivers.sv	bfm_drivers.sv links the class library to the BFMs in testbench.qsys.
	defines.sv	Edit defines.sv to vary quantities such as the number of pixels in parallel and the latency on the Avalon ST bus.
	nios_control_model.sv	CSR accesses to the VIP cores in the DUT are made using nios_control_model.sv in the same way that a Nios processor would in hardware.
	run.tcl	run.tcl is called by both tests. Edit for other simulators if needed.
	testbench.qsys	testbench.qsys instances dut.qsys together with a BFM for every Avalon-ST and Avalon-MM interface.
	testbench_stimulus.sv	testbench_stimulus.sv instances a testbench.v file (created when Platform Designer generates a simulation model from testbench.qsys) and includes all the other test files required. It is the top level of hierarchy in the simulations.

A.2.5.1. bfm_drivers.sv

For every BFM that is instanced in testbench.qsys, there must be a systemVerilog BFM driver object declared. The declaration handles all the interfacing between the BFM and the class library.

The declaration methodology in `bfm_drivers.sv` follows a specific template. The format below shows an example of the Avalon-ST sink BFM code in the `bfm_drivers.sv` file:

```
1 `define SINK st_sink_bfm_0
2 `define SINK_STR "st_sink_bfm_0"
3 `define SINK_HIERARCHY_NAME `TESTBENCH.`SINK
4 `include "av_st_video_sink_bfm_class.sv"
5 `define CLASSNAME c_av_st_video_sink_bfm_`SINK
6 `CLASSNAME `SINK;
7 `undef CLASSNAME
8
9 mailbox #(c_av_st_video_item) m_video_items_for_src_bfm[1][1];
10
11 initial
12 begin
13 m_video_items_for_sink_bfm[0][0] = new(0);
14 st_sink_bfm_0  = new(m_video_items_for_sink_bfm[0]);
15 end
```

The code template breaks down as follows.

In lines 1-2, the name of the sink BFM in the Platform Designer testbench is defined, together with its hierarchical path in the netlist in line 3. (The `av_st_video_sink_bfm_class.sv` code requires the name of the sink and hierarchy to be defined prior to the code being included. The prior definition ensures that the driver can find the BFM in the design hierarchy.

A bespoke class is declared and an object with the sink name instanced in line 6.

In line 9, a 1x1 mailbox array of video items is declared—the first element of which is constructed in line 13.

In line 14, the 1D mailbox element array is passed to constructor of the new sink object—this is the mechanism by which video items (video packet, control packets and user packets) are passed to the user's test code.

A similar methodology is used for Avalon-MM interface. The example below shows a code that can be found at the beginning of the `bfm_drivers.sv` file:

```
1 `define IF_MEM_MASTER_RD
2 `define SLAVE_NAME mm_slave_bfm_for_vfb_reads
3 `define SLAVE_HIERARCHICAL_LOCATION testbench.mm_slave_bfm_for_vfb_reads
4 `include "av_mm_slave_bfm_class_inc.sv"
5 av_mm_slave_bfm_mm_slave_bfm_for_vfb_reads #(32, 32/8, 0)
slave_bfm_mm_slave_bfm_for_vfb_reads;
6 mailbox #(av_mm_transaction #(32,  32/8, 0)) mbox_slave_bfm_mem_master_rd_drv;
7 mailbox #(av_mm_transaction #(32,  32/8, 0))
mbox_slave_bfm_mem_master_rd_reply_drv;
8 initial begin
9 mbox_slave_bfm_mem_master_rd_drv = new(0);
10 mbox_slave_bfm_mem_master_rd_reply_drv = new(0);
11 slave_bfm_mm_slave_bfm_for_vfb_reads = new(mbox_slave_bfm_mem_master_rd_drv,
mbox_slave_bfm_mem_master_rd_reply_drv);
12 end
13 `undef IF_MEM_MASTER_RD
14 `undef SLAVE_NAME
15 `undef SLAVE_HIERARCHICAL_LOCATION
```

In lines 5 and 6, mailboxes of control register objects are declared and constructed.

The control register class has an address field and a value field together with other information. These mailboxes are passed to the control BFM class in line 8, and are used to send register writes and to receive register reads, as described in the following section.

A.2.5.2. nios_control_model.sv

The register items mailboxes defined in the BFM drivers file are used in the `testbench/nios_control_model.sv` by creating a new control register object, setting the address and data as required, and passing it to the mailbox, as shown in the **send_write_to_mixer** task code:

```
task send_write_to_mixer(int unsigned address, int unsigned data);
    automatic c_av_mm_control_register register_update = new(1);
    register_update.use_event = 0;
    register_update.write = 1;
    register_update.address = address;
    register_update.value = data;
    m_register_items_for_mixer_control_bfm.put(register_update);
endtask
```

The `nios_control_model.sv` code contains the writes to the Mixer II and Video Frame Buffer II IP cores, as required by both the example constrained random and video files tests:

```
`ifdef CONSTRAINED_RANDOM_TEST
send_write_to_mixer(8, 0); // X offset
repeat (10) @ (posedge (av_st_clk));
$display("%t NIOS CONTROL EMULATOR : Test harness writing 0 to the X offset of
the Mixer",$time);

send_write_to_mixer(9, 0); // Y offset
repeat (10) @ (posedge (av_st_clk));
$display("%t NIOS CONTROL EMULATOR : Test harness writing 0 to the Y offset of
the Mixer",$time);

send_write_to_mixer(3, `MIXER_BACKGROUND_WIDTH_SW);
repeat (10) @ (posedge (av_st_clk));
$display("%t NIOS CONTROL EMULATOR : Test harness writing a background layer of
width %0d to the Mixer",$time,`MIXER_BACKGROUND_WIDTH_SW);

send_write_to_mixer(4, `MIXER_BACKGROUND_HEIGHT_SW);
repeat (10) @ (posedge (av_st_clk));
$display("%t NIOS CONTROL EMULATOR : Test harness writing a background layer of
height %0d to the Mixer",$time,`MIXER_BACKGROUND_HEIGHT_SW);

`else
send_write_to_mixer(8, 20); // X offset
repeat (10) @ (posedge (av_st_clk));
$display("%t NIOS CONTROL EMULATOR : Test harness writing 20 to the X offset of
the Mixer",$time);

send_write_to_mixer(9, 20); // Y offset
repeat (10) @ (posedge (av_st_clk));
$display("%t NIOS CONTROL EMULATOR : Test harness writing 20 to the Y offset of
the Mixer",$time);
`endif
```

The control interfaces for each VIP core in the DUT has their own BFM driver and associated mailbox.

To port these tasks and test register accesses to C code for a Nios 2 processor to run on real hardware, you need to factor in the base addresses for each VIP IP core in the DUT.

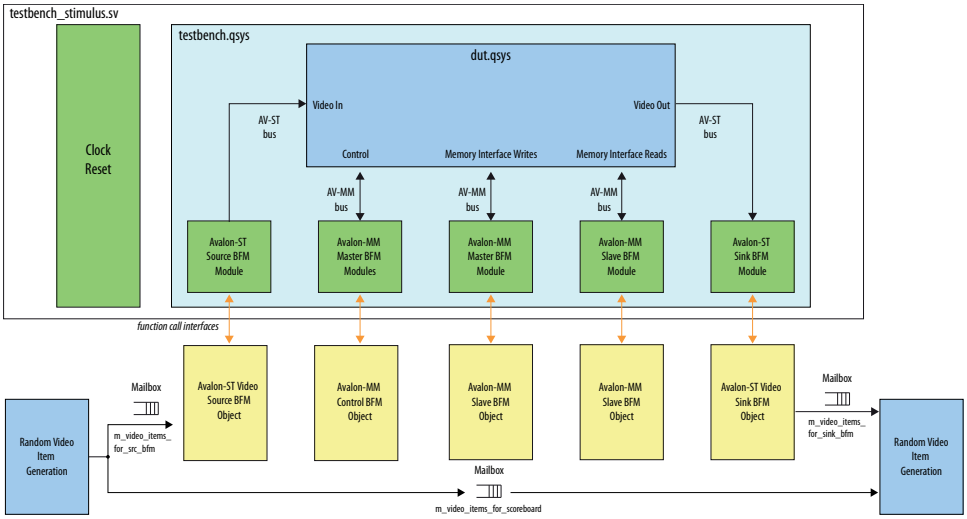
A.2.6. Constrained Random Test

The constrained random test example is easily assembled using the class library.

Note: The steps to run the constrained random are described at the start of this section.

Figure 92. Example of a Constrained Random Test Environment

The figure below shows the constrained random test environment structure.



Randomized video, control, and user packets are generated using the SystemVerilog's built-in constrained random features. The DUT processes the video packets and the scoreboard determines a test pass or fail result.

`test.sv` comprises the following code sections:

Declaration	Description
1 Object Declarations	

`18 c_av_st_video_control #(`BITS_PER_SYMBOL, `CHANNELS_PER_PIXEL) video_control_pkt1, video_control_pkt2;`
`19 c_av_st_video_control #(`BITS_PER_SYMBOL, `CHANNELS_PER_PIXEL) video_control_dut, video_control_golden;`
`20 c_av_st_video_data #(`BITS_PER_SYMBOL, `CHANNELS_PER_PIXEL) video_data_dut, video_data_scoreboard;`
`21 c_av_st_video_data #(`BITS_PER_SYMBOL, `CHANNELS_PER_PIXEL) dut_video_pkt, scoreboard_video_pkt;`
`22 c_av_st_video_data video_data_real_pkt1, video_data_real_pkt2 ;`
`23 c_av_st_video_item dut_pkt;`
`24 c_av_st_video_item scoreboard_pkt;`
	Declare some objects to be used later in the test. Notice that the number of pixels in parallel is not needed in these definitions as that is abstracted away from the user and handled via the classes in `av_st_video_source_bfm_class.sv` and `av_st_video_sink_bfm_class.sv`.
	Note that the video item class is the base class, so could hold objects of any of the more specialized classes, such as control or video data.

2 Mailbox Declarations	Description

`27 mailbox #(c_av_st_video_item) m_video_items_for_scoreboard[1];`
`28`
`29 int input_frame_count = 0;`
`30`
`31 initial`
`32 begin`
`33`
	The `bfm_drivers.sv` file contains mailbox declarations required for transmission and receipt of video packets, but the test harness requires a mailbox for the scoreboard. It is declared in line 27 and constructed at time zero in line 35.

2 Mailbox Declarations	Description
34 // Construct the 0th element of the video items mailbox array 35 m_video_items_for_scoreboard[0] = new(0);	

3 Start the Source and Sink BFMs	Description
61 #0 // Delta-cycle delay to ensure the BFM drivers were constructed in bfm_drivers.sv 62 fork // .start() calls to the VIP BFM drivers are blocking. Fork the calls. 63 st_source_bfm_0.start(); 64 st_sink_bfm_0.start(av_st_clk); 65 join_none;	The source and sink BFMs must be started. Once started, the source will wait for objects for sending to be pushed into its mailbox whilst the sink will push objects into its mailbox when one is received from the DUT.

4 Stimulus—Start Producing Control and Video Packets	Description
109 produce_control_pkt(add_to_scoreboard,`MAX_WIDTH,`MAX_HEIGHT,quantization,ctrl_width,ctrl_height,ctrl_interlacing); 110 111 // Don't add video packet to scoreboard if a drop packet will be IMMEDIATELY following it 112 if (`COLOR_PLANES_ARE_IN_PARALLEL == 1) 113 produce_video_pkt((ctrl_width), ctrl_height, add_to_scoreboard); 114 else 115 produce_video_pkt((ctrl_width*`CHANNELS_PER_PIXEL), ctrl_height, add_to_scoreboard);	In the main loop calls are made to the tasks in `class_library/tasks.sv` to produce randomly sized control packets with matching video packets.

4 Packet Checker—Checking Produced Control and Video Packets Against the Scoreboard	Description
161 while ((op_pkts_seen <= (input_frame_count*2)) && !timeout) 162 begin : main_checker_loop // NB x2 for control packets 163 164 @(posedge(av_st_clk)); 165 166 if (m_video_items_for_sink_bfm[0][0].try_get(dut_pkt) == 1) 167 begin : packet_found	The corresponding loop to the stimulus loop is the checker code. This polls the sink's mailbox for video items, categorizes according to type and to which packet type is expected, compares video packets to those previously input and then marks the packet as pass or fail accordingly. Line 161 shows that the checker loop runs until twice the input frame count has been seen (control plus video packet per input frame) and uses the `try_get()` call on the sink mailbox in line 166 to check for DUT packets.
174 // DUT has output a Video packet 175 if (dut_pkt.get_packet_type() != control_packet) 176 begin : possible_video_packet_found 177 178 if (dut_pkt.get_packet_type() == video_packet) 179 begin : video_packet_found 180 181 dut_video_pkt = c_av_st_video_data'(dut_pkt); //' 182 183 // First output packets should be colour bars : 184 if ((op_pkts_seen <= `MIXER_NUM_TPG_OUTPUT_FRAMES*2) && !expecting_dut_control_packet) 185 begin : absorb_colour_bars 186 $display("%t DUT DOUT : PASS - DUT output an assumed mixer background pattern video packet %0d(VIDEO length == %0d).",$time,op_pkts_seen,dut_video_pkt.get_length()); 187 end 188 189 else 190 begin : mixed_packets 191 192 // Sledgehammer approach for example test - search all input frames for the one currently output: 193 matching_frame = 0; 194 frames_to_check = m_video_items_for_scoreboard[0].num(); 195 while (frames_to_check>0 && !matching_frame) begin 196 frames_to_check--;	Once the packet type is determined (lines 174-179) it is cast to an object of the corresponding type (see line 181). As earlier described, in `testbench/nios2_control_model.sv`, the Mixer II is set up to offset the output video from the frame buffer by zero pixels for constrained random testing, so line 184 below shows that we expect to see one frame of test pattern (while the frame buffer receives and buffers the first input frame) followed by the video packets. The code from lines 195-205 iterates through every item in the scoreboard, casting to a video packet type as required, before comparing to the DUT video packet in line 200 (using the built-in `silent_compare` method from `class_library/av_st_video_classes`, which does not generate output if the packets do not match). A flag is set if a match is found and the scoreboard packet is pushed onto a temporary mailbox store. After the search, all scoreboard packets are popped off the temporary store ready for comparing with subsequent video frames. Control packets are compared against a golden one in a similar way.

continued...

4 Packet Checker—Checking Produced Control and Video Packets Against the Scoreboard	Description
197 m_video_items_for_scoreboard[0].get(scoreboard_pkt); 198 if (scoreboard_pkt.get_packet_type() == video_packet) begin 199 scoreboard_video_pkt = c_av_st_video_data'(scoreboard_pkt); //' 200 if (dut_video_pkt.silent_compare(scoreboard_video_pkt) == 1) 201 matching_frame = 1; 202 203 m_tmp.put(scoreboard_pkt); 204 end 205 end	

A.3. Complete Class Reference

A.3.1. c_av_st_video_control

The declaration for the *c_av_st_video_control* class:

```
class c_av_st_video_control #(parameter BITS_PER_CHANNEL = 8,
CHANNELS_PER_PIXEL = 3) extends c_av_st_video_item;
```

Table 90. Method Calls for c_av_st_video_control Class

Method Call	Description
function new();	
function bit compare (c_av_st_video_control r);	Compares this instance to object r. Returns 1 if identical, 0 if otherwise.
function void copy(c_av_st_video_control #(BITS_PER_CHANNEL, CHANNELS_PER_PIXEL) c);	Copies the control packet object argument into the object making the method call.
function bit [15:0] get_width ();	—
function void set_min_width(int width);	This is the minimum width for control packets produced using calls to the randomize() method.
function void set_max_width(int width);	This is the minimum width for control packets produced using calls to the randomize() method.
function void set_min_height(int height);	This is the minimum width for control packets produced using calls to the randomize() method.
function void set_max_height(int height)	This is the minimum width for control packets produced using calls to the randomize() method.
function bit [15:0] get_height ();	—
function bit [3:0] get_interlacing ();	—
function t_packet_control get_append_garbage ();	—
function int get_garbage_probability ();	—
function void set_width (bit [15:0] w);	—
function void set_height (bit [15:0] h);	—
function void set_interlacing (bit [3:0] i);	—

continued...

Method Call	Description
`function void set_append_garbage (t_packet_control i);`	Refer to `append_garbage` member.
`function void set_garbage_probability (int i);`	—
`function string info();`	Returns a formatted string containing the width, height and interlacing members.

Table 91. Members of c_av_st_video_control Class

Member	Description
`rand bit[15:0] width;`	—
`rand bit[15:0] height;`	—
`rand bit[3:0] interlace;`	—
`int min_width = 30;`	—
`int max_width = 200;`	—
`int min_height = 30;`	—
`int max_height = 200;`	—
`int length;`	—
`rand t_packet_control append_garbage = off;`	The `append_garbage` control is of type `t_packet_control`, defined as: `typedef enum{on,off,random} t_packet_control;`
`rand int garbage_probability = 50;`	The source BFM uses `garbage_probability` and `append_garbage` to determine whether or not to append garbage beats to the end of the control packets. Garbage beats are generated with a probability of `Garbage_probability`%. • When a stream of garbage is being generated, the probability that the stream terminates is fixed in the source BFM at 10%. • When garbage is produced, this typically produces around 1 to 30 beats of garbage per control packet.

A.3.2. c_av_st_video_data

The declaration for the *c_av_st_video_data* class:

```
class c_av_st_video_data#(parameter BITS_PER_CHANNEL = 8,
CHANNELS_PER_PIXEL = 3) extends c_av_st_video_item;
```

Table 92. Method Calls for c_av_st_video_data Class

Method Call	Description
`function new();`	Constructor
`function void copy (c_av_st_video_data c);`	Copies object c into this object.
`function bit compare (c_av_st_video_data r);`	Compares this instance to object r. Returns 1 if identical, 0 if otherwise.

continued...

Method Call	Description
`function bit silent_compare (c_av_st_video_data #(BITS_PER_CHANNEL,CHANNELS_PER_PIXEL) r);`	Identical to compare, but creates no output messages.
`function bit is_same_size (c_av_st_video_data #(BITS_PER_CHANNEL,CHANNELS_PER_PIXEL) r);`	Returns 1 if two video packets are the same length, 0 if otherwise.
`function bit visualise (int height, int width);`	For small frames (<100x100) a table of hex values is produced.
`function bit make_test_pattern (int height, int width, int spacing);`	Populates the pixels in a frame with a color-bars style test pattern (3 color planes only)
`function bit make_random_field (int height, int width);`	Populates the pixels in a frame with random colors.
`function void set_height(int h);`	—
`function int get_height();`	—
`function void set_width(int w);`	—
`function int get_width();`	—
`function void set_min_length(int length);`	—
`function int get_length();`	—
`function void set_line_length(int line_length);`	Line length is used to split the frame into lines for transmission.
`function int get_line_length();`	—
`function void set_max_length(int length);`	—
`function int get_length();`	—
`function c_pixel #(BITS_PER_CHANNEL,CHANNELS_PER_PIXEL) pop_pixel();`	Returns a pixel object from the packet in first in first out (FIFO) order.
`function c_pixel #(BITS_PER_CHANNEL,CHANNELS_PER_PIXEL) query_pixel(int i);`	Returns a pixel object from the packet at index i, without removing the pixel.
`function void unpopulate(bit display);`	Pops all pixels from the packet, displaying them if display = 1.
`function void delete_pixel(int index);`	Removes pixel "index" from the frame.
`function void push_pixel(c_pixel #(BITS_PER_CHANNEL, CHANNELS_PER_PEXEL)pixel);`	Pushes a pixel into the packet.

Table 93. Members of c_av_st_video_data Class

Member	Description
`c_pixel #(BITS_PER_CHANNEL,CHANNELS_PER_PIXEL)pixels [$];`	The video data is held in a queue of pixel objects.
`c_pixel #(BITS_PER_CHANNEL,CHANNELS_PER_PIXEL) pixel, new_pixel, r_pixel;`	Pixel objects used for storing intermediate data.

continued...

Member	Description
`rand int video_length;`	The length of the video packet (used for constrained random generation only).
`int video_max_length = 10;`	Maximum length of video packet (used for constrained random generation only).
`int video_min_length = 5;`	Minimum length of video packet (used for constrained random generation only).

A.3.3. c_av_st_video_file_io

The declaration for the *c_av_st_video_file_io* class:

```
class c_av_st_video_file_io#(parameter BITS_PER_CHANNEL = 8,
CHANNELS_PER_PIXEL = 3);
```

Table 94. Method Calls for c_av_st_video_file_io Class

Method Call	Description
`function void set_send_control_packets(t_packet_controls);`	If this method is used to set the `send_control_packet` control to off, then one control packet is sent at the beginning of video data, but no further control packets are sent.
`function t_packet_control get_send_control_packets();`	—
`function void set_send_user_packets(t_packet_control s);`	If the `send_user_packets` control is off, no user packets at all are sent. Otherwise, user packets are sent before and after any control packets.
`function t_packet_control get_send_user_packets();`	—
`function void set_send_early_eop_packets(t_packet_control s);`	If the send_eop_packets control is off, all packets are of the correct length (or longer). Otherwise, early EOP are sent of a length determined by the constraints on `early_eop_packet_length`.
`function t_packet_control get_send_early_eop_packets();`	—
`function void set_early_eop_probability(int s);`	If the `send_early_eop_packets` control is set to random, the `early_eop_probability` control determines what proportion of video packets are terminated early.
`function int get_early_eop_probability();`	—
`function void set_send_late_eop_packets(t_packet_controls);`	If the `send_late_eop_packets` control is off, all packets are of the correct length (or longer). Otherwise, late EOP are sent of a length determined by the constraints on `late_eop_packet_length`.
`function t_packet_control get_send_late_eop_packets();`	—
`function void set_late_eop_probability (int s);`	If the `send_late_eop_packets` control is set to random, the `late_eop_probability` control determines what proportion of video packets are terminated late.
`function int get_late_eop_probability ();`	—

continued...

Method Call	Description
`function void set_user_packet_probability (int s);`	If the `send_user_packets` is set to random, the `user_packet_probability` control determines the probability that a user packet being sent before a control packet. It also determines the probability that a user packet will be sent after a control packet.
`function int get_user_packet_probability ();`	—
`function void set_control_packet_probability(int s);`	If the `send_control_packets` control is set to random, the `control_packet_probability` control determines the probability of a control packet being sent before a video packet.
`function int get_control_packet_probability();`	—
`function void set_send_garbage_after_control_packets (t_packet_control s);`	When the `send_control_packet()` method puts a control packet into the `m_video_item_out` mailbox, the `append_garbage` member of the control packet object is set to the value of `send_garbage_after_control_packets`.
`function t_packet_control get_send_garbage_after_control_packets();`	—
`function void set_object_name(string s);`	You can use `object_name` to name a given object instance of a class to ensure any reporting that the class generates is labeled with the originating object's name.
`function string get_object_name();`	—
`function string get_filename();`	This returns the filename associated with the object, by the `open_file` call.
`function void set_image_height(bit[15:0]height);`	—
`function bit[15:0]get_image_height();`	—
`function void set_image_width(bit[15:0] width);`	—
`function bit[15:] get_image_width();`	—
`function void set_video_data_type(string s);`	Sets the `fourcc[3]` code associated with the raw video data. The following are the supported four character code (FOURCC) codes: • `RGB32` • `IYU2` • `YUY2` • `Y410` • `A2R10GB10` • `Y210`
`function string get_video_data_type();`	Returns the FOURCC code (for example, RGB32) being used for the raw video data.
`function int get_video_packets_handled();`	—
`function new(mailbox #(c_av_st_video_item)m_vid_out);`	Constructor. The mailbox is used to pass all packets in and out of the file I/O object.
`function void open_file(string fname, t_rwrw);`	Files are opened using this method. For example: `video_file_reader.open_file(''vip_car_0.bin", read);` `t_rw` is an enumerated type with values read or write.

continued...

Method Call	Description
	NB. The read fails if there is no associated `.spc` file, for example, `vip_car_o.spc`).
`function void close_file();`	For example, `video_file_reader.close_file();`
`task read_file();`	`Read_file()` optionally calls `send_user_packet()` and `send_control_packet()`, then calls `read_video_packet()`.
`task send_control_packet();`	The control packet sent is derived from the image height, width, and interlace fields as provided by `open_file()`.
`task send_user_packet();`	The user packet sent is always comprised of random data and had a maximum length hard-coded to 33 data items.
`task_generate_spc_file();`	When writing a file, this call creates the necessary associated `.spc` file.
`task read_video_packet();`	The main file reading method call. Binary data is read from the file and packed into pixel objects according to the settings of `ycbr_pixel_order` and endianism. Pixel objects are packed into a video data object, with some pixels optionally added or discarded if late/early EOP is being applied. When one complete field of video has been read (as determined by the height and width controls), the `video_data` object is put in the mailbox.
`task wait_for_and_write_video_packet_to_file();`	When called, this method waits for an object to be put in the mailbox (usually from a sink BFM). When a control or a user packet object arrives, this call is reported and ignored. When a video packet arrives, the video data is written to the open file in little endianism format.

Table 95. Members of c_av_st_video_file_io Class

Member	Description
`local int video_packets_handled = 0;`	`video_packets_handled` is added whenever a packet is read or written to or from the file.
`local int control_packets_handled = 0;`	`control_packets_handled` is added whenever a control packet is put in the object's mailbox.
`local int user_packets_handled = 0;`	`user_packets_handled` is added whenever a user packet is put in the object's mailbox.
`local reg[15:0] image_height;`	—
`local reg[15:0] image_width;`	—
`local reg[3:0] image_interlaced;`	—
`string image_fourcc;`	—
`local string object_name = "file_io";`	—
`local string filename;`	—
`local string spc_filename;`	—
`local int fourcc_channels_per_pixel;`	Set when the associate `.spc` file is read.
`local int fourcc_bits_per_channel;`	Set when the associate `.spc` file is read.

continued...

Member	Description
`local int fourcc_pixels_per_word;`	Set when the associate `.spc` file is read.
`local int fourcc_channel_lsb;`	Set when the associate `.spc` file is read.
`int early_eop_probability = 20;`	—
`Int late_eop_probability = 20;`	—
`int user_packet_probability = 20;`	—
`int control_packet_probability = 20;`	—
`mailbox #(c_av_st_video_item) m_video_item_out = new(0);`	The mailbox is used to pass all packets in/out of the file i/o object.
`rand t_packet_control send_control_packets = on;`	—
`rand t_packet_control send_user_packets = off;`	—
`rand t_packet_control send_early_eop_packets = off;`	—
`rand t_packet_control send_late_eop_packets = off;`	If both `send_late_eop_packets` and `send_early_eop_packets` are set to random, a late EOP will only be generated if an early EOP has not been generated.
`rand t_packet_control send_garbage_after_control_packets = off;`	—
`rand int early_eop_packet_length = 20;`	`constraint early_eop_length { early_eop_packet_length dist {1:= 10, [2:image_height*image_width-1]:/90}; early_eop_packet_length inside {[1:image_height*image_width]}; }`
`rand int late_eop_packet_length = 20;`	`constraint late_eop_length { late_eop_packet_length inside {[1:100]}; }`

A.3.4. c_av_st_video_item

The declaration for the *c_av_st_video_item* class:

```
class c_av_st_video_item;
```

Table 96. Method Calls for c_av_st_video_item Class

Method Call	Description
`function new();`	Constructor
`function void copy (c_av_st_video_item c);`	Sets `this.packet_type` to match that of c.
`function void set_packet_type (t_packet_types ptype);`	—
`function t_packet_types get_packet_type();`	—

Table 97. Members of c_av_st_video_item Class

Member	Description
`t_packet_types packet_type;`	`Packet_type` must be one of the following:

Member	Description
	• video_packet • control_packet • user_packet • generic_packet • undefined

A.3.5. c_av_st_video_source_sink_base

The declaration for the *c_av_st_video_source_sink_base* class:

```
class c_av_st_video_source_sink_base;
```

Table 98. Method Calls for c_av_st_video_source_sink_base Class

Method Call	Description
function new(mailbox #(c_av_st_video_item)m_vid);	Constructor. The video source and sink classes transfer video objects through their mailboxes.
function void set_readiness_probability(int percentage);	—
function int get_readiness_probability();	—
function void set_long_delay_probability(real percentage);	—
function real get_long_delay_probability();	—
function void set_long_delay_duration_min_beats(int percentage);	—
function int get_long_delay_duration_min_beats();	—
function void set_long_delay_duration_max_beats(int percentage);	—
function int get_long_delay_duration_max_beats();	—
function void set_pixel_transport(t_pixel_format in_parallel);	—
function t_pixel_format get_pixel_transport();	—
function void set_name(string s);	—
function string get_name();	—

Table 99. Members of c_av_st_video_source_sink_base Class

Member	Description
mailbox # (c_av_st_video_item) m_video_items= new(0);	The Avalon-ST video standard allows you to send symbols in serial or parallel format. You can set this control to either format.
t_pixel_format pixel_transport = parallel;	—
string name = "undefined";	—
int video_packets_sent = 0;	—
int control_packets_sent = 0;	—

continued...

Member	Description
`int user_packets_sent = 0;`	—
`int readiness_probability = 80;`	Determines the probability of when a sink or source is ready to receive or send data in any given clock cycle, as manifested on the bus by the READY and VALID signals, respectively.
`real long_delay_probability = 0.01;`	• The `readiness_probability` control provides a *steady state* readiness condition. • The `long_delay_probability` allows for the possibility of a much rarer and longer period of unreadiness, of durations of the order of the raster line period or even field period.
`rand int long_delay_duration_min_beats= 100;`	This control sets the minimum duration (as measured in data beats of) a long delay. *Note:* If `pixel_transport` = parallel than one data beats = one pixel = one clock cycle.
`rand int long_delay_duration_max_beats = 1000;`	This control sets the maximum duration (as measured in data beats) of a long delay.
`rand int long_delay_duration = 80;`	`constraint c1 {long_delay_duration inside [long_delay_duration_min_beats: long_delay_duration_max_beats]};}`

A.3.6. c_av_st_video_sink_bfm_'SINK

The declaration for the *c_av_st_video_sink_bfm_'SINK* class:

```
'define CLASSNAME c_av_st_video_sink_bfm_'SINK;
 class 'CLASSNAME extends c_av_st_video_source_sink_base;
```

Table 100. Method Calls for c_av_st_video_sink_bfm_'SINK Class

This class does not have additional members to those of the base class.

Method Call	Description
`function new(mailbox#(c_av_st_video_item)m_vid);`	Constructor.
`task start;`	The start method simply waits until the reset of the Avalon-ST sink BFM goes inactive, then calls the `receive_video()`task.
`task receive_video;`	The `receive_video` task continually drives the Avalon-ST sink BFM's ready signal in accordance with the probability settings in the base class. It also continually captures `signal_received_transaction` events from the Avalon-ST sink BFM and uses the Avalon-ST sink BFM API to read bus data. Bus data is decoded according to the Avalon-ST video specification and data is packed into an object of the appropriate type (video, control or, user). The object is then put into the mailbox.

A.3.7. c_av_st_video_source_bfm_'SOURCE

The declaration for the *c_av_st_video_source_bfm_'SOURCE* class:

```
'define CLASSNAME c_av_st_video_source_bfm_'SOURCE
 class 'CLASSNAME extends c_av_st_video_source_sink_base;
```

Table 101. Method Calls for c_av_st_video_source_bfm_'SOURCE Class

This class does not have additional members to those of the base class.

Method Call	Description
`function new(mailbox#(c_av_st_video_item)m_vid)`	Constructor.
`task start;`	The start method simply waits until the reset of the Avalon-ST source BFM goes inactive, then continually calls the `send_video()` task.
`task send_video;`	The `send_video()` task waits until a video item is put into the mailbox, then it drives the Avalon-ST sink BFM's API accordingly. The `set_transaction_idles()` call is used to set the valid signal in accordance with the probability settings in the base class. The mailbox object is categorized according to object type. Each object is presented to the bus according to the Avalon-ST Video specification and the setting of the `pixel_transport` control.

A.3.8. c_av_st_video_user_packet

The declaration for the *c_av_st_video_user_packet* class:

```
class c_av_st_video_user_packet#(parameters BITS_PER_CHANNEL=8,
CHANNELS_PER_PIXEL=3) extends c_av_st_video_item;
```

Table 102. Method Calls for c_av_st_video_user_packet Class

Method Call	Description
`function new();`	Constructor.
`function void copy (c_av_st_video_user_packet c);`	Copies object c into this object.
`function bit compare (c_av_st_video_user_packet r);`	Compares this instance to object r. Returns 1 if identical, 0 if otherwise.
`function void set_max_length(int l);`	Minimum and maximum lengths are used for constrained random testing only.
`function void set_min_length(int l);`	Minimum and maximum lengths are used for constrained random testing only.
`function int get_length();`	—
`function bit[3:0] get_identifier();`	The identifier is the Avalon-ST video packet identifier. 0x0 indicates video, 0xf indicates a control packet, and the user packets take random values from 0x4 to 0xe.
`function bit[3:0] set_identifier(int i);`	—
`function bit [BITS_PER_CHANNEL*CHANNELS_PER_PIXEL-1:0] pop_data();`	Returns the next beat of user data.
`function bit [BITS_PER_CHANNEL*CHANNELS_PER_PIXEL-1:0] query_data(int i);;`	Returns the next beat of user data without removing it from the object.
`function void push_data(bit [BITS_PER_CHANNEL*CHANNELS_PER_PIXEL-1:0] d);`	—

Table 103. Members of c_av_st_video_user_packet Class

Member	Description
`rand bit[BITS_PER_CHANNEL*CHANNELS_PER_PIXEL-1:0]data[$]`	User data is stored as a queue of words.
`rand bit[3:0] identifier;`	`constraint c2 {identifier inside {[1:9]};}`
`int max_length = 10;`	`constraint c1 { data.size() inside {[min_length:max_length]}; data.size() % pixels_in_parallel == 0; }`
`int min_length = 5;`	`constraint c1 { data.size() inside {[min_length:max_length]}; data.size() % pixels_in_parallel == 0; }`

A.3.9. c_pixel

The declaration for the *c_pixel* class:

```
class c_pixel#(parameters BITS_PER_CHANNEL=8, CHANNELS_PER_PIXEL=3);
```

Table 104. Method Calls for c_pixel Class

Method Call	Description
`function new();`	Constructor.
`function void copy(c_pixel #(BITS_PER_CHANNEL, CHANNELS_PER_PIXEL) pix);`	Copies object pixel into this object.
`function bit[BITS_PER_CHANNEL-1:0] get_data(int id);`	Returns pixel data for channel id.
`function void set_data(int id, bit [BITS_PER_CHANNEL-1:0] data);`	Sets pixel data for channel id.

A.3.10. av_mm_transaction

The *av_mm_transaction* class is the base class for an Avalon-MM transaction (command), including member variables for address, data, read or write, and byte enable values.

Table 105. Method Calls for av_mm_transaction Class

Method Call	Description
`function new(int max_args);`	Class constructor. `max_args` is only used during the randomization process as a maximum number of arguments. It does not limit how many arguments you can assign to the class.
`function void copy(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH , USE_BYTE_ENABLE) val_to_cpy);`	Deep copy from `val_to_cpy` into class.
`function cmd get_cmd();`	Returns the current command (`READ` or `WRITE`).
`function void set_cmd(Cmd cmd);`	Sets the current command (`READ` or `WRITE`).
`function longint get_address();`	Returns the read/write address.

continued...

Method Call	Description
`function void set_address(longint addr);`	Sets the read/write address.
`function int num_args();`	Returns the number of arguments (1 argument corresponds to 1 data beat on the bus).
`function void set_num_args(int length);`	Sets the number of arguments.
`function int byte_enable_length();`	Sets the number of byte enables – used for writes whose bytes are not modulo BYTE_WIDTH in size.
`function void print_config_one_arg_per_line(bit one_arg_per_line);`	If called with an argument of 1, any following calls to `ps_printf` will print the details for each argument (beat on the bus) on a new line.
`function string ps_printf(string indent_each_line_with);`	Prints the entire command.
`function bit args_match(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE) cmp);`	Compares *cmp* against the *av_mm_transaction* object and returns a 1 if the number of arguments and value of each argument in a message match. Called by the *equals* method call.
`function bit byte_enable_match(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE) cmp);`	Compares the byte enable values in *cmp* against the *av_mm_transaction* object's byte enable values and returns a 1 if the number of byte enables and values match. Called by the *equals* method call.
`function bit equals(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE) cmp);`	Compares *cmp* against the *av_mm_transaction* object and returns a 1 if they are identical.

Table 106. Members of av_mm_transaction Class

Member	Description
`int c_max_args;`	Maximum number of arguments for constrained random generation.
`rand enum { READ, WRITE } cmd;`	The command itself (read or write).
`rand bit [ADDR_WIDTH-1 : 0] addr;`	Address.
`rand bit [BYTE_WIDTH*8-1 : 0] arguments [$];`	Arguments / data beats.
`rand byte byte_enable [$];`	Byte enable bus values.

A.3.11. av_mm_master_bfm_`MASTER_NAME

The declaration for the *av_mm_master_bfm_`MASTER_NAME* class:

```
`define IF_NAME(name)`"name`"
`define create_classname(__prefix, __suffix) __prefix``__suffix
`define AV_MM_MASTER_CLASSNAME `create_classname(av_mm_master_bfm_, `MASTER_NAME)
class `AV_MM_MASTER_CLASSNAME  #(int ADDR_WIDTH = 32, int BYTE_WIDTH = 4, bit
USE_BYTE_ENABLE = 1'b0 );
```

Table 107. Method Calls for av_mm_master_bfm_`MASTER_NAME Class

Method Call	Description
`function new(mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) mbox_transaction_in, mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) mbox_read_reply_out);`	Constructor.
`function mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) get_transaction_mbox_handle();`	Returns the *transaction_in_mailbox* handle, to allow transactions to be sent to the master.
`function mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) get_read_reply_mbox_handle();`	Returns the *read_reply_out_mailbox* handle, to allow transaction read replies to be returned from the master.
`function void set_response_timeout(input int new_response_timeout);`	–
`function void set_command_timeout(input int new_command_timeout);`	–
`task start();`	Starts polling the *transaction_in_mailbox* for commands (which are then transferred to the BFM), and then starts polling the BFM response queue for pending responses, which are then posted in the *read_reply_out_mailbox*.

Table 108. Members of av_mm_master_bfm_`MASTER_NAME Class

Member	Description
`mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) transaction_in_mailbox;`	–
`mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) read_reply_out_mailbox;`	–
`mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) internal_mailbox;`	–

A.3.12. av_mm_slave_bfm_`SLAVE_NAME

The declaration for the *av_mm_slave_bfm_`SLAVE_NAME* class:

```
`define IF_NAME(name)`"name`"
`define create_classname(__prefix, __suffix) __prefix``__suffix
`define AV_MM_SLAVE_CLASSNAME `create_classname(av_mm_slave_bfm_, `SLAVE_NAME)
class `AV_MM_SLAVE_CLASSNAME  #(int ADDR_WIDTH = 32, int BYTE_WIDTH = 4, bit
USE_BYTE_ENABLE = 1'b0 );
```

Table 109. Method Calls for av_mm_slave_bfm_`SLAVE_NAME Class

Method Call	Description
`function new(mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) mbox_transaction_out, mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) mbox_read_reply_in);`	Constructor.
`function mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) get_transaction_mbox_handle();`	Returns the *transaction_in_mailbox* handle, to allow transactions to be sent to the master.
`function mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) get_read_reply_mbox_handle();`	Returns the *read_reply_out_mailbox* handle, to allow transaction read replies to be returned from the master.
`function void set_response_timeout(input int new_response_timeout);`	–
`task start();`	Starts polling the *read_reply_in_mailbox* for read data (which is then transferred from the BFM), and then starts polling the BFM command queue for pending commands, which are then posted in the *transaction_out_mailbox*.

Table 110. Members of av_mm_slave_bfm_`SLAVE_NAME Class

Member	Description
`mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) transaction_out_mailbox;`	–
`mailbox #(av_mm_transaction #(ADDR_WIDTH, BYTE_WIDTH, USE_BYTE_ENABLE)) read_reply_in_mailbox;`	–

A.3.13. av_mm_control_register

This class is instanced by objects of type *c_av_mm_base_class* to contain address and data information about a control register. The declaration for the *av_mm_control_register* class:

```
class c_av_mm_control_register;
```

Table 111. Method Calls for av_mm_ control_register Class

Method Call	Description
`function new(int unsigned no_of_triggers);`	Constructor.
`function void copy(c_av_mm_control_register c);`	–

Table 112. Members of av_mm_ control_register Class

Member	Description
`bit write;`	–
`int unsigned address;`	–
`int unsigned value;`	–

continued...

Member	Description
int unsigned init_latency;	–
bit use_event;	–
event trigger[];	–
bit trigger_enable[];	–

A.3.14. av_mm_control_base

The declaration for the *av_mm_control_base* class:

```
class c_av_mm_control_base #(int unsigned AV_ADDRESS_W = 32, int unsigned
AV_DATA_W = 32);
```

Table 113. Method Calls for av_mm_ control_base Class

Method Call	Description
function new(mailbox #(c_av_mm_control_register) m_register, mailbox #(c_av_mm_control_register) m_readdata);	Constructor.
function void set_name(string s);	–
function string get_name();	–
function void set_idleness_probability(int percentage);	Constrained random only.

Table 114. Members of av_mm_ control_base Class

Member	Description
string name = "undefined";	–
mailbox #(c_av_mm_control_register) m_register_items;	–
mailbox #(c_av_mm_control_register) m_readdata;	–

A.4. Data Format

Figure 93. Supported FOURCC Codes and Data Format

The figure below shows an example of the data format required by the file I/O class for each of the supported FOURCC codes.