Video and Image Processing Suite

User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Software Version: 9.0
Document Date: March 2009

UG-VIPSUITE-8.0

Contents

Appendix A. Reference

1. About This MegaCore Function Suite

New Features

This release supports the following new features:

- The Deinterlacer MegaCore function supports controlled frame dropping or repeating to keep the input and output frame rates locked together.
- The Test Pattern Generator MegaCore function can generate a user-specified constant color that can be used as a uniform background.
- Added preliminary support for Arria® II GX devices.

Release Information

Table 1–1 provides information about this release of the Altera® Video and Image Processing Suite MegaCore® functions.

Table 1–1. Video and Image Processing Suite Release Information

Item	Description	
Version	9.0 (All MegaCore functions)	
Release Date	March 2009	
Ordering Code	IPS-VIDEO (Video and Image Processing Suite)	
Product IDs	0003 (Color Space Converter) 00B1 (Chroma Resampler) 00B2 (Gamma Corrector) 00B3 (2D FIR Filter) 00B4 (2D Median Filter) 00B5 (Alpha Blending Mixer) 00B6 (Deinterlacer) 00C3 (Frame Buffer)	00B7 (Scaler) 00B8 (Line Buffer Compiler) 00C8 (Clipper) 00C4 (Clocked Video Input) 00C5 (Clocked Video Output) 00C9 (Color Plane Sequencer) 00CA (Test Pattern Generator)
Vendor ID(s)	6AF7	

For more information about this release, refer to the *MegaCore IP Library Release Notes and Errata*.

Device Family Support

MegaCore functions provide either full or preliminary support for target Altera device families:

- *Full support* means the MegaCore function meets all functional and timing requirements for the device family and may be used in production designs.

- *Preliminary suppor*t means the MegaCore function meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution.

Table 1–2 shows the level of support offered by the Video and Image Processing Suite MegaCore functions to each Altera device family.

Table 1–2. Device Family Support

Device Family	Support
Arria® II GX	Preliminary
Arria GX	Full
Cyclone® II	Full
Cyclone III	Full
HardCopy® II	Full
Stratix®	Full
Stratix II	Full
Stratix II GX	Full
Stratix III	Full
Stratix IV	Preliminary
Stratix GX	Full
Other device families	No support

Features

The following key fetaures are common to all of the Video and Image Processing Suite MegaCore functions:

- Common Avalon® Streaming (Avalon-ST) interface and Avalon-ST Video protocol
- Avalon Memory-Mapped (Avalon-MM) interfaces for run-time control input and connections to external memory blocks
- Easy-to-use MegaWizard™ interface for parameterization and hardware generation
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Support for OpenCore Plus evaluation
- SOPC Builder ready

General Description

The Altera Video and Image Processing Suite is a collection of MegaCore functions that facilitate the development of video and image processing designs. The MegaCore functions are suitable for use in a wide variety of image processing and display applications.

Color Space Converter

The Color Space Converter MegaCore function transforms video data between color spaces. These color spaces allow you to specify colors using three coordinate values. The MegaCore function supports some pre-defined conversions between standard color spaces, and allows the entry of custom coefficients to translate between any two three-valued color spaces. You can configure this MegaCore function to change conversion values at run time using an Avalon-MM slave interface.

Chroma Resampler

The Chroma Resampler MegaCore function resamples video data to and from common sampling formats. The human eye is more sensitive to brightness than it is to 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 sampling formats that specify how this subsampling is done are part of the MPEG-1, MPEG-2, H.261 and other standards.

Gamma Corrector

The Gamma Corrector MegaCore function allows video streams to be corrected for the physical properties of display devices. For example, the brightness displayed by a cathode-ray tube monitor has a non-linear response to the voltage of a video signal. To account for this response, you can program the MegaCore function with a look-up table that models the non-linear function. The look-up table is then used to transform the video data and give the best image on the display.

2D FIR Filter

The 2D FIR Filter MegaCore function performs 2D convolution using matrices of 3×3, 5×5, or 7×7 coefficients. The MegaCore function retains full precision throughout the calculation while making efficient use of FPGA resources. With suitable coefficients, the MegaCore function can perform operations such as sharpening, smoothing, and edge detection. You can configure this MegaCore function to change coefficient values at run time using an Avalon-MM slave interface.

2D Median Filter

The 2D Median Filter MegaCore function provides a means to apply 3×3, 5×5, or 7×7 pixel median filters to video images. Median filtering removes speckle noise and salt-and-pepper noise while preserving the sharpness of edges in video images.

Alpha Blending Mixer

The Alpha Blending Mixer MegaCore function can mix together up to 12 image layers. The MegaCore function supports both picture-in-picture mixing and image blending. Each image layer can be independently activated and moved at run time using an Avalon-MM slave interface.

Scaler

The Scaler MegaCore function provides a means to resize video streams. The MegaCore function supports nearest-neighbor, bilinear, bicubic, and polyphase scaling algorithms. You can configure this MegaCore function to change resolutions and/or filter coefficients at run time using an Avalon-MM slave interface.

Clipper

The Clipper MegaCore function provides a means to clip video streams. You can configure this MegaCore function at compile time or optionally at run time using an Avalon-MM slave interface.

Deinterlacer

The Deinterlacer MegaCore function converts interlaced video to progressive video using a bob, weave, or simple motion-adaptive algorithm. 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.

Additionally, the Deinterlacer MegaCore function can provide double -buffering or triple-buffering in external RAM. Double-buffering can help solve throughput problems (burstiness) in video systems. Triple-buffering can provide simple frame rate conversion.

Frame Buffer

The Frame Buffer MegaCore function buffers video frames into external RAM. The MegaCore function supports double or triple-buffering with a range of options for frame dropping and repeating.

Line Buffer Compiler

The Line Buffer Compiler MegaCore function efficiently maps video line buffers to Altera on-chip memories.

The Line Buffer Compiler MegaCore function is not available in SOPC Builder.

Clocked Video Input

The Clocked Video Input MegaCore function converts clocked video formats (such as BT656 and DVI) to Avalon-ST Video. You can configure this MegaCore function at run time using an Avalon-MM slave interface.

Clocked Video Output

The Clocked Video Output MegaCore function converts Avalon-ST Video to clocked video formats (such as BT656 and DVI). You can configure this MegaCore function at run time using an Avalon-MM slave interface.

Color Plane Sequencer

The Color Plane Sequencer MegaCore function changes how color plane samples are transmitted across the Avalon-ST interface.

You can configure the channel order in sequence or in parallel. In addition to reordering color plane samples, this MegaCore function splits and joins video streams, giving control over the routing of color plane samples.

Test Pattern Generator

The Test Pattern Generator generates a video stream that displays either still color bars for use as a test pattern or a constant color for use as a uniform background. You can use this MegaCore function during the design cycle to validate a video system without the possible throughput issues associated with a real video input.

Example Design

An example design is available that illustrates the cost, performance, and quality capabilities of Video and Image Processing MegaCore functions in the Altera design flow.

For more information about this example design, refer to *AN427: Video and Image Processing Up Conversion Example Design*.

MegaCore Verification

Before releasing a version of each MegaCore function, Altera runs comprehensive regression tests to verify quality and correctness.

Custom variations of the MegaCore functions are generated to exercise various parameter options. The resulting simulation models are thoroughly simulated and the results verified against bit-accurate master simulation models.

Performance and Resource Utilization

This section shows typical expected performance for the Video and Image Processing Suite MegaCore functions when using the Quartus II with Cyclone III and Stratix III devices.

Cyclone III devices use combinational look-up tables (LUTs) and logic registers; Stratix III devices use combinational adaptive look-up tables (ALUTs) and logic registers.

Color Space Converter

Table 1–3 shows the performance figures for the Color Space Converter.

Table 1–3. Color Space Converter Performance

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Converting 1,080 pixel 10-bit Studio R'G'B' to HDTV Y'CbCr using 18-bit coefficients and 27-bit summands.							
Cyclone III *(1)*	517	592	—	—	6	—	237
Stratix III *(2)*	430	505	—	—	—	6	350
Converting 1024×768 14-bit Y'UV to Computer R'G'B' using 18-bit coefficients and 15-bit summands.							
Cyclone III *(1)*	525	633	—	—	6	—	237
Stratix III *(2)*	421	537	—	—	—	6	329
Converting 640×480 8-bit SDTV Y'CbCr to Computer R'G'B' using 9-bit coefficients and 16-bit summands, color planes in parallel.							
Cyclone III *(1)*	574	818	—	—	9	—	238
Stratix III *(2)*	469	665	—	—	—	9	394
Converting 720×576 8-bit Computer R'G'B' to Y'UV using 9-bit coefficients and 8-bit summands.							
Cyclone III *(1)*	394	427	—	—	3	—	238
Stratix III *(2)*	337	376	—	—	—	3	395

Notes to Table 1–3:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

Chroma Resampler

Table 1–4 shows the performance figures for the Chroma Resampler.

Table 1–4. Chroma Resampler Performance (Part 1 of 2)

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Upsampling from 4:2:0 to 4:4:4 with a parallel data interface and run time control of resolutions up to extended graphics array format (XGA - 1024x768). This parameterization uses luma-adaptive filtering on the horizontal resampling and nearest-neighbor on the vertical resampling.							
Cyclone III *(1)*	2,262	1,771	16,384	4	—	—	158
Stratix III *(2)*	1,559	1,769	16,384	4	—	—	261
Upsamping from 4:2:2 to 4:4:4 with a sequential data interface at quarter common intermediate format (QCIF - 176x144) using luma-adaptive filtering.							
Cyclone III *(1)*	998	787	—	—	—	—	212
Stratix III *(2)*	656	785	—	—	—	—	356
Downsampling from 4:4:4 to 4:2:0 with a parallel data interface and run-time control of resolutions up to XGA (1024x768). The parameterization uses anti-aliasing filtering on the horizontal resampling and nearest-neighbor on the vertical.							
Cyclone III *(1)*	1,848	1,236	4,096	1	—	—	149
Stratix III *(2)*	1,115	1,240	4,096	1	—	—	296

Table 1–4. Chroma Resampler Performance (Part 2 of 2)

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Downsamping from 4:4:4 to 4:2:2 with a sequential data interface at quarter common intermediate format (QCIF - 176x144) using an anti-aliasing filter.							
Cyclone III *(1)*	848	531	—	—	—	—	194
Stratix III *(2)*	419	533	—	—	—	—	357

Notes to Table 1–4:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

Gamma Corrector

Table 1–5 shows the performance figures for the Gamma Corrector.

Table 1–5. Gamma Corrector Performance

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Gamma correcting 1,080 pixel one color 10-bit data.							
Cyclone III *(1)*	242	153	10,260	3	—	—	233
Stratix III *(2)*	172	153	10,260	3	—	—	393
Gamma correcting 720×576 one color 10-bit data.							
Cyclone III *(1)*	242	153	10,260	3	—	—	233
Stratix III *(2)*	172	153	10,260	3	—	—	393
Gamma correcting 128×128 three color 8-bit data.							
Cyclone III *(1)*	226	137	2,064	1	—	—	229
Stratix III *(2)*	160	137	2,064	1	—	—	383
Gamma correcting 64×64 three color 8-bit data.							
Cyclone III *(1)*	226	137	2,064	1	—	—	229
Stratix III *(2)*	160	137	2,064	1	—	—	383

Notes to Table 1–5:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

2D FIR Filter

Table 1–6 on page 1–8 shows the performance figures for the 2D FIR Filter.

Table 1–6. 2D FIR Filter Performance

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Edge detecting 3×3 asymmetric filter, working on 352×288 8-bit R'G'B', using 3 bit coefficients							
Cyclone III *(1)*	965	987	16,896	4	9	—	190
Stratix III *(2)*	750	830	16,896	4	—	9	349
Smoothing 3×3 symmetric filter, working on 640×480 8-bit R'G'B', using 9 bit coefficients.							
Cyclone III *(1)*	981	960	30,720	4	6	—	195
Stratix III *(2)*	761	909	30,720	4	—	4	354
Sharpening 5×5 symmetric filter, working on 640×480 in 8-bit R'G'B', using 9 bit coefficients.							
Cyclone III *(1)*	1,858	1,791	61,440	8	12	—	183
Stratix III *(2)*	1,398	1,598	61,440	8	—	8	295
Smoothing 7×7 symmetric filter, working on 1,280×720 in 10-bit R'G'B', using 15 bit coefficients							
Cyclone III *(1)*	3,584	3,612	230,400	30	20	—	164
Stratix III *(2)*	2,663	3,365	230,400	30	—	20	263

Notes to Table 1–6:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

2D Median Filter

Table 1–7 shows the performance figures for the 2D Median Filter.

Table 1–7. 2D Median Filter Performance

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
3×3 median filtering HDTV 720 pixel monochrome video.							
Cyclone III *(1)*	1,575	1,200	25,600	6	—	—	233
Stratix III *(2)*	994	1,200	25,600	6	—	—	351
Median filtering 64×64 pixel R'G'B frames using a 3×3 kernel of pixels.							
Cyclone III *(1)*	1,535	1,154	3,072	2	—	—	230
Stratix III *(2)*	971	1,155	3,072	2	—	—	349
Median filtering 352×288 pixel two color frames using a 5×5 kernel of pixels.							
Cyclone III *(1)*	5,416	3,828	28,160	8	—	—	203
Stratix III *(2)*	2,682	3,832	28,160	8	—	—	300
7×7 median filtering 352×288 pixel monochrome video.							
Cyclone III *(3)*	10,813	7,296	16,896	6	—	—	191
Stratix III *(2)*	4,850	7,296	16,896	6	—	—	270

Notes to Table 1–7:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

(3) Using EP3C16F484C6 devices.

Alpha Blending Mixer

Table 1–8 shows the performance figures for the Alpha Blending Mixer.

Table 1–8. Alpha Blending Mixer Performance

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Alpha blending an on-screen display within a region of 1,024×768 pixel 10-bit Y'CbCr 4:4:4 video. Alpha blending is performed using 16 levels of opacity from fully opaque to fully translucent.							
Cyclone III *(1)*	1,103	764	752	1	4	—	178
Stratix III *(2)*	797	733	752	1	—	3	319
Drawing a picture-in-picture window over the top of a 128×128 pixel background image in 8-bit R'G'B' color.							
Cyclone III *(1)*	735	492	8,432	3	—	—	136
Stratix III *(2)*	609	548	752	1	—	—	354
Rendering two images over 352×240 pixel background 8-bit R'G'B' video.							
Cyclone III *(1)*	1,207	760	752	1	—	—	189
Stratix III *(2)*	853	758	752	1	—	—	325
Using alpha blending to composite three layers over the top of PAL resolution background video in 8-bit monochrome. Alpha blending is performed using 256 levels of opacity from fully opaque to fully translucent.							
Cyclone III *(1)*	1,924	1,300	752	1	6	—	169
Stratix III *(2)*	1,428	1,205	752	1	—	6	276

Notes to Table 1–8:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

Scaler

Table 1–9 shows the performance figures for the Scaler.

Table 1–9. Scaler Performance (Part 1 of 2)

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Scaling 640×480, 8-bit, three color data up to 1,024×768 with linear interpolation. This can be used to convert video graphics array format (VGA - 640×480) to video electronics standards association format (VESA - 1024×768).							
Cyclone III *(1)*	945	687	30,720	6	4	—	191
Stratix III *(2)*	682	624	30,720	6	—	4	346
Scaling R'G'B' QCIF to common intermediate format (CIF) with no interpolation.							
Cyclone III *(1)*	434	297	4,224	3	—	—	223
Stratix III *(2)*	304	298	4,224	3	—	—	393
Scaling up or down between NTSC standard definition and 1080 pixel high definition using 10 taps horizontally and 9 vertically. Resolution and coefficients are set by a run-time control interface.							
Cyclone III *(3)*	3,842	3,095	417,456	58	19	—	161
Stratix III *(2)*	2,225	2,757	417,456	58	—	19	264

Table 1–9. Scaler Performance (Part 2 of 2)

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Scaling NTSC standard definition (720x480) RGB to high definition 1080p using a bicubic algorithm.							
Cyclone III *(1)*	1,687	1,185	69,444	13	8	8	169
Stratix III *(2)*	1,039	1,050	69,444	14	—	8	294

Notes to Table 1–9:

(1) Using EP3C10F256C6 devices.
(2) Using EP3SE50F780C2 devices.
(3) Using EP3C40F780C6 devices.

Clipper

Table 1–10 shows the performance figures for the Clipper.

Table 1–10. Clipper Performance

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9x9)	(18x18)	
A 1080p60-compatible clipper with a clipping window that has fixed offsets from the size of the input frames.							
Cyclone III *(1)*	649	484	—	—	—	—	202
Stratix III *(2)*	475	484	—	—	—	—	332
A 100×100 pixel clipper with a clipping window that is a rectangle from the input frames.							
Cyclone III *(1)*	416	275	—	—	—	—	192
Stratix III *(2)*	323	276	—	—	—	—	333
A 1080p60-compatible clipper with a runtime interface which uses offsets to set the clipping window.							
Cyclone III *(1)*	819	619	—	—	—	—	189
Stratix III *(2)*	597	620	—	—	—	—	327
A 100×100 pixel clipper with a run-time interface which uses a rectangle to set the clipping window.							
Cyclone III *(1)*	560	468	—	—	—	—	207
Stratix III *(2)*	449	468	—	—	—	—	326

Notes to Table 1–10:

(1) Using EP3C10F256C6 devices.
(2) Using EP3SE50F780C2 devices.

Deinterlacer

Table 1–11 shows the performance figures for the Deinterlacer.

Table 1–11. Deinterlacer Performance (Part 1 of 2)

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory			DSP Blocks		f_{MAX} (MHz)
			ALUTs	Bits	M9K	(9×9)	(18×18)	
Deinterlacing 64×64 pixel 8-bit R'G'B' frames using the bob algorithm with scanline duplication.								
Cyclone III *(1)*	538	292	—	1,536	1	—	—	189
Stratix III *(2)*	386	293	—	1,536	1	—	—	325

Table 1–11. Deinterlacer Performance (Part 2 of 2)

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory			DSP Blocks		f_{MAX} (MHz)
			ALUTs	Bits	M9K	(9×9)	(18×18)	
Deinterlacing with scanline interpolation using the bob algorithm working on 352×288 pixel 12-bit Y'CbCr 4:2:2 frames.								
Cyclone III *(1)*	673	395	—	8,448	2	—	—	184
Stratix III *(2)*	492	398	—	8,448	2	—	—	312
Deinterlacing PAL (720×576) with 8-bit Y'CbCr 4:4:4 color using the motion-adaptive algorithm.								
Cyclone III *(3)*	5,723	5,678	—	81,514	39	1	—	121
Stratix III *(4)*	4,803	5,772	5	73,292	41	—	1	243
Deinterlacing HDTV 1080i resolution with 12-bit Y'CbCr 4:4:4 color using the weave algorithm.								
Cyclone III *(1)*	3,231	2,546	—	3,078	15	—	—	170
Stratix III *(2)*	3,539	2,540	—	3,078	19	—	—	280

Notes to Table 1–11:

(1) Using EP3C10F256C6 devices.
(2) Using EP3SE50F780C2 devices.
(3) Using EP3C40F780C6 devices.
(4) Using EP3SE110F1152C2 devices.

Frame Buffer

Table 1–12 shows the performance figures for the Frame Buffer.

Table 1–12. Frame Buffer Performance

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Double-buffering XGA (1024×768) 8-bit RGB with a sequential data interface.							
Cyclone III *(1)*	2,103	1,725	8,408	6	—	—	161
Stratix III *(2)*	1,749	1,726	8,432	11	—	—	300
Triple-buffering VGA (640×480) 8-bit RGB with a parallel data interface.							
Cyclone III *(1)*	2,121	1,670	7,368	6	—	—	169
Stratix III *(2)*	1,737	1,671	7,368	10	—	—	290
Triple-buffering VGA (640×480) 8-bit RGB buffering up to 32 large Avalon-ST Video packets into RAM.							
Cyclone III *(1)*	3,796	2,495	11,168	6	—	—	145
Stratix III *(2)*	2,723	2,496	11,168	11	—	—	228
Triple-buffering 720×576 8-bit RGB with sequential data interface and runtime control interface.							
Cyclone III *(1)*	2,177	1,763	8,504	7	—	—	162
Stratix III *(2)*	1,826	1,763	8,504	12	—	—	304

Notes to Table 1–12:

(1) Using EP3C10F256C6 devices.
(2) Using EP3SE50F780C2 devices.

Line Buffer Compiler

Table 1–13 shows the performance figures for the Line Buffer Compiler.

Table 1–13. Line Buffer Compiler Performance

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Two 10-bit wide 1,920 pixel lines. This configuration could be used in an HDTV 1080i system.							
Cyclone III *(1)*	40	38	38,400	5	—	—	357
Stratix III *(2)*	43	38	38,400	5	—	—	655
Two 8-bit wide 720 pixel lines. This configuration would be appropriate for 8-bit PAL line buffering.							
Cyclone III *(1)*	34	32	12,288	2	—	—	357
Stratix III *(2)*	38	32	12,288	2	—	—	696
Three 8-bit wide 64 pixel lines.							
Cyclone III *(1)*	20	20	1,536	1	—	—	357
Stratix III *(2)*	21	20	1,536	1	—	—	717
Four 10-bit wide 1,280 pixel lines. This parameterization might be used for 10-bit HDTV 720 pixel line buffering.							
Cyclone III *(1)*	33	32	51,200	7	—	—	357
Stratix III *(2)*	36	32	51,200	7	—	—	655

Notes to Table 1–13:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

Clocked Video Input

Table 1–14 shows the performance figures for the Clocked Video Input.

Table 1–14. Clocked Video Input Performance (Part 1 of 2)

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory				f_{MAX} (MHz)
			ALUTs	M9K	Bits	MLAB Bits	
Converts DVI 1080p60 clocked video to Avalon-ST Video.							
Cyclone III *(1)*	411	414	—	7	51,200	—	187
Stratix III *(2)*	264	414	—	7	51,200	—	245
Converts PAL clocked video to Avalon-ST Video.							
Cyclone III *(1)*	417	417	—	3	22,528	—	183
Stratix III *(2)*	301	417	—	3	22,528	—	228
Converts SDI 1080i60 clocked video to Avalon-ST Video.							
Cyclone III *(1)*	417	439	—	7	43,028	—	169
Stratix III *(2)*	319	458	10	6	43,028	40	226

Table 1–14. Clocked Video Input Performance (Part 2 of 2)

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory				f_{MAX} (MHz)
			ALUTs	M9K	Bits	MLAB Bits	
Converts SDI 1080p60 clocked video to Avalon-ST Video.							
Cyclone III *(1)*	414	430	—	7	43,008	—	174
Stratix III *(2)*	292	458	10	6	43,008	40	226

Notes to Table 1–14:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

Clocked Video Output

Table 1–15 shows the performance figures for the Clocked Video Output.

Table 1–15. Clocked Video Output Performance

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory				f_{MAX} (MHz)
			ALUTs	M9K	Bits	MLAB Bits	
Converts Avalon-ST video to DVI 1080p60 clocked video.							
Cyclone III *(1)*	261	221	—	7	51,200	—	191
Stratix III *(2)*	174	221	—	7	51,200	—	287
Converts Avalon-ST video to PAL clocked video.							
Cyclone III *(1)*	279	207	—	3	22,528	—	212
Stratix III *(2)*	213	207	—	3	22,528	—	317
Converts Avalon-ST video to SDI 1080i60 clocked video.							
Cyclone III *(1)*	294	216	—	6	43,008	—	199
Stratix III *(2)*	230	216	—	6	43,008	—	301
Converts Avalon-ST video to SDI 1080p60 clocked video.							
Cyclone III *(1)*	295	216	—	6	43,008	—	200
Stratix III *(2)*	229	216	—	6	43,008	—	271

Notes to Table 1–15:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

Color Plane Sequencer

Table 1–16 shows the performance figures for the Color Plane Sequencer.

Table 1–16. Color Plane Sequencer Performance (Part 1 of 2)

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Rearranging a channels in sequence 4:2:2 stream, from Cb,Y,Cr,Y to Y,Cb,Y,Cr. 8 bit data.							
Cyclone III *(1)*	291	243	—	—	—	—	261
Stratix III *(2)*	204	243	—	—	—	—	436

Table 1–16. Color Plane Sequencer Performance (Part 2 of 2)

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Joining a single channel luminance stream and a channels in sequence horizontally half-subsampled chrominance stream to a single 4:2:2 channels in sequence output stream. 8 bit data.							
Cyclone III *(1)*	374	313	—	—	—	—	261
Stratix III *(2)*	262	313	—	—	—	—	391
Splitting a 4:2:2 stream from 2 channels in parallel to a single channel luminance output stream and a channels in sequence horizontally half-subsampled chrominance output stream. 8 bit data.							
Cyclone III *(1)*	451	335	—	—	—	—	231
Stratix III *(2)*	305	336	—	—	—	—	369
Rearranging 3 channels in sequence to 3 channels in parallel. 8 bit data.							
Cyclone III *(1)*	242	249	—	—	—	—	258
Stratix III *(2)*	186	253	—	—	—	—	440

Notes to Table 1–16:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

Test Pattern Generator

Table 1–16 shows the performance figures for the Test Pattern Generator

Table 1–17. Test Pattern Generator Performance

Device Family	Combinational LUTs/ALUTs	Logic Registers	Memory		DSP Blocks		f_{MAX} (MHz)
			Bits	M9K	(9×9)	(18×18)	
Producing a 400×x200, 8-bit 4:2:0 Y'Cb'Cr' stream with a parallel data interface.							
Cyclone III *(1)*	164	112	192	2	—	—	332
Stratix III *(2)*	158	113	192	2	—	—	545
Producing a 640×480, 8-bit R'G'B' stream with a sequential data interface.							
Cyclone III *(1)*	212	118	192	3	—	—	287
Stratix III *(2)*	181	119	192	3	—	—	485
Producing a 720×480, 10-bit 4:2:2 Y'Cb'Cr' interlaced stream with a sequential data interface.							
Cyclone III *(1)*	258	138	240	3	—	—	245
Stratix III *(2)*	233	138	240	3	—	—	489
Producing a 1920×1080, 10-bit 4:2:2 Y'Cb'Cr' interlaced stream with a parallel data interface. The resolution of the pattern can be changed using the run-time control interface.							
Cyclone III *(1)*	365	208	304	4	—	—	263
Stratix III *(2)*	248	208	304	4	—	—	481

Notes to Table 1–17:

(1) Using EP3C10F256C6 devices.

(2) Using EP3SE50F780C2 devices.

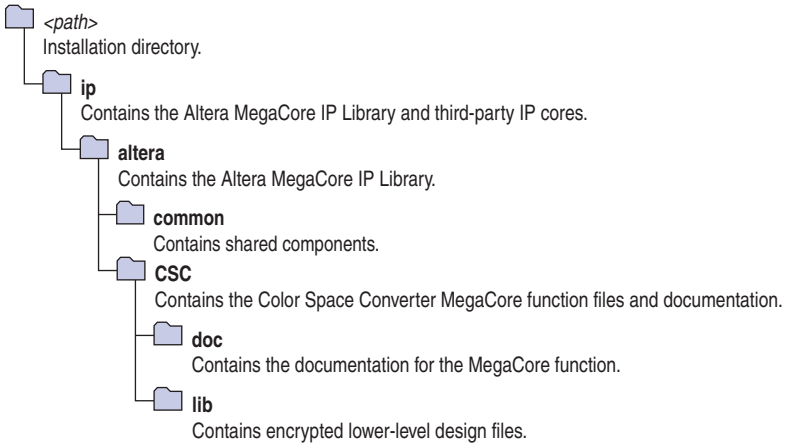
Installation and Licensing

The Video and Image Processing Suite MegaCore functions are included in the Altera MegaCore IP Library, which is a part of the Altera Complete Design Suite. To install a MegaCore function, install the MegaCore IP Library.

For system requirements and installation instructions, refer to *Quartus II Installation & Licensing for Windows and Linux Workstations*.

Figure 1–1 shows the directory structure for a typical Video and Image Processing Suite MegaCore function where <*path*> is the installation directory for the Quartus® II software.

Figure 1–1. Directory Structure



The default installation directory on Windows is **c:\altera\<*version*>**; or on Linux is **/opt/altera<*version*>**.

OpenCore Plus Evaluation

You can use Altera's free OpenCore Plus evaluation feature to evaluate the MegaCore function in simulation and in hardware before you purchase a license. You can perform the following actions:

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP^{SM} megafunction) in your system.
- Verify the functionality of your design, and quickly evaluate its size and speed.
- Generate time-limited device programming files for designs that include MegaCore functions.
- Program a device and verify your design in hardware.

You must purchase a license for the MegaCore function only when you are completely satisfied with its functionality and performance, and want to take your design to production.

After you purchase a license, you can request a license file from the Altera website at www.altera.com/licensing and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative

For more information on OpenCore Plus hardware evaluation using MegaCore functions, refer to *AN 320: OpenCore Plus Evaluation of Megafunctions*.

OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation supports the following modes of operation:

- Untethered—the design runs for a limited time.
- Tethered—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely.

All megafunctions in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior might be masked by the time-out behavior of the other megafunctions.

The untethered time-out for all Video and Image Processing Suite MegaCore functions is one hour; the tethered time-out value is indefinite. The `reset` signal is forced high when the hardware evaluation time expires. This keeps the Video and Image Processing MegaCore function permanently in its reset state.

Design Flows

The Video and Image Processing Suite MegaCore functions support the following design flows:

- **SOPC Builder**: Use this flow if you want to create an SOPC Builder system that includes a Video and Image Processing Suite MegaCore function variation. The SOPC Builder flow supports a wide range of connectable components including direct memory access (DMA) controllers, on-chip memories, Nios® II processor and other components with Avalon Memory-Mapped (Avalon-MM) or Avalon Streaming (Avalon-ST) interfaces.
- **MegaWizard™ Plug-In Manager**: Use this flow if you want to create a Video and Image Processing Suite MegaCore function variation that you can instantiate manually in your design.

This chapter describes how you can use a Video and Image Processing Suite MegaCore function in each of these flows. The parameterization available for each MegaCore function is similar and is described in "Parameter Settings" on page 3–1.

The Line Buffer Compiler MegaCore function is available only in the MegaWizard Plug-In Manager flow.

After parameterizing and simulating a design in either of these flows, you can compile and program the completed design in the Quartus II software.

SOPC Builder Flow

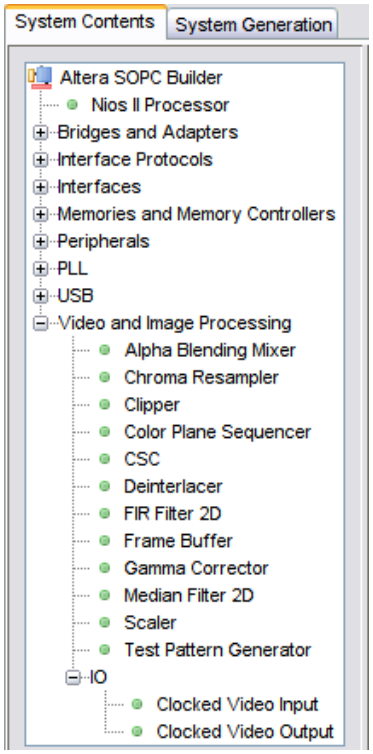
SOPC Builder is a Quartus II software tool that enables you to rapidly and easily build systems and evaluate embedded systems. The SOPC Builder flow allows you to add a Video and Image Processing Suite MegaCore function directly to a new or existing SOPC Builder system. You can add components to create an SOPC Builder system with other components such as a Nios II processor and external memory controller. SOPC Builder automatically creates the system interconnect logic and system simulation environment.

The following steps describe how you can use Video and Image Processing Suite MegaCore functions in the SOPC Builder flow:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. Launch **SOPC Builder** from the Tools menu.
3. For a new system, specify the system name and language.
4. Add and parameterize any required memory and processor modules to your system from the **System Contents** tab.
5. Add the required Video and Image Processing Suite MegaCore function to your system from the Video and Image Processing section in the **System Contents** tab.

Figure 2–1 shows the SOPC Builder **System Contents** tab.

Figure 2–1. Video and Image Processing Components in SOPC Builder



1. Use the MegaWizard interface to specify the required parameters for the MegaCore function variation. For information about the parameters available for each MegaCore function, refer to Chapter 3, Parameter Settings.
2. Click **Finish** in the MegaWizard interface to complete the parameterization and add it to the system.
3. Connect the components using the SOPC Builder patch panel.
4. If you intend to simulate your SOPC builder system, select **Simulation** on the **System Generation** tab and click **Generate** to generate a functional simulation model for the system.

   For information about the generated files, refer to "Generated Files" on page 2–6.

For more information about the SOPC Builder flow, refer to volume 4 of the *Quartus II Handbook*.

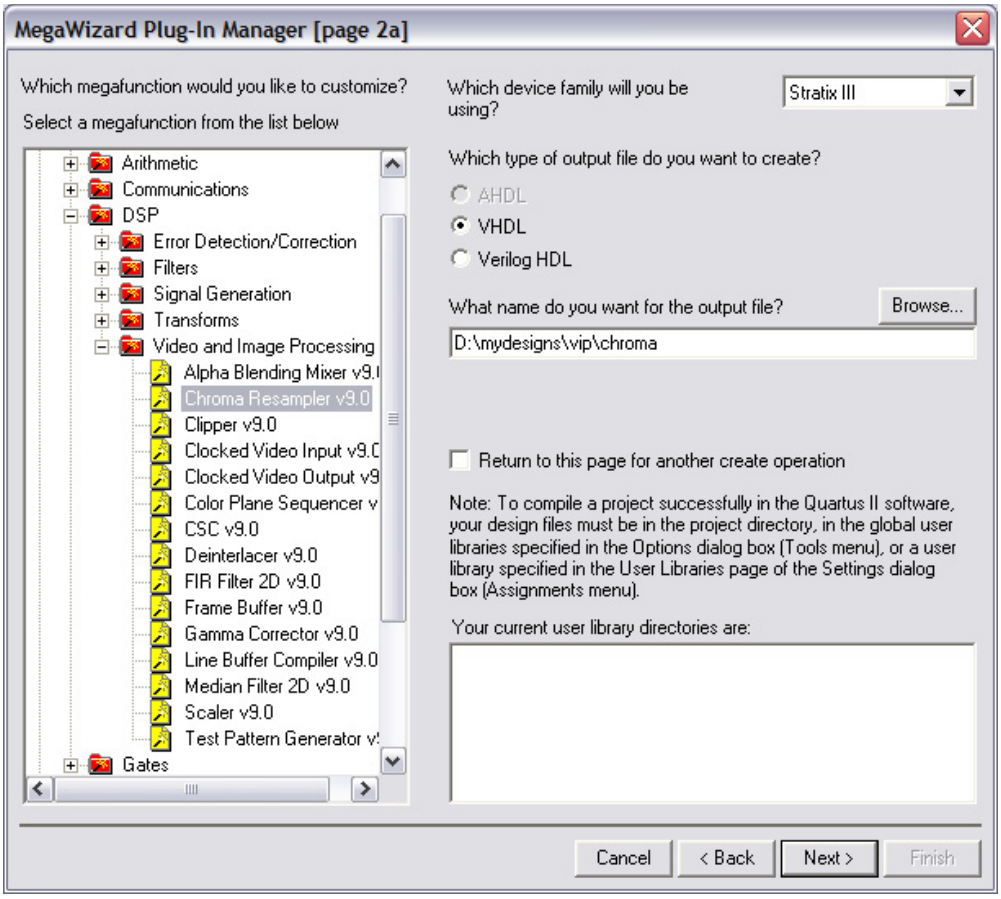
MegaWizard Plug-in Manager Flow

The MegaWizard Plug-in Manager flow allows you to customize a Video and Image Processing Suite MegaCore function, and manually integrate the MegaCore function variation in a Quartus II design.

The following steps describe how you can use Video and Image Processing Suite MegaCore functions in the MegaWizard Plug-in Manager flow:

1. Create a new project using the **New Project Wizard** available from the File menu in the Quartus II software.
2. Launch **MegaWizard Plug-in Manager** from the Tools menu, and select the option to create a new custom megafunction variation.
3. Click **Next** and select the required MegaCore function from the DSP Video and Image Processing section in the **Installed Plug-Ins** tab (Figure 2–2).

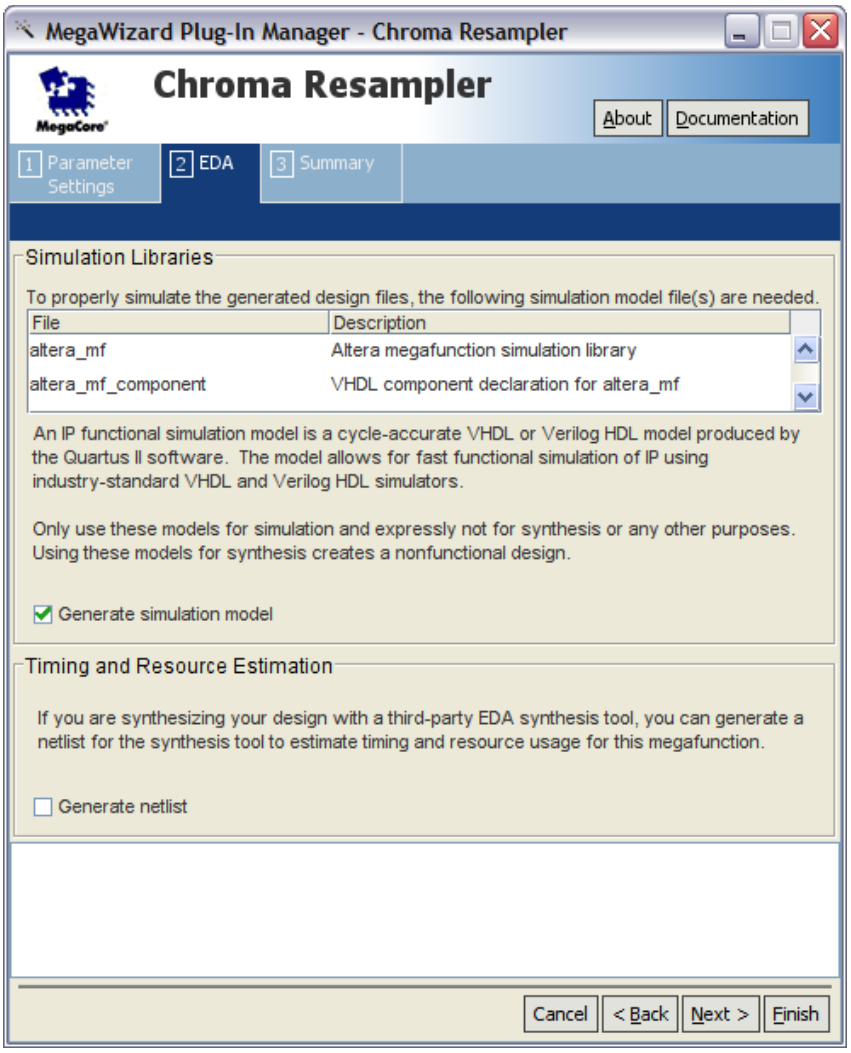
Figure 2–2. Selecting the MegaCore Function



4. Verify that the device family is the same as you specified in the **New Project Wizard**.
5. Select the top-level output file type for your design; the wizard supports VHDL and Verilog HDL.
6. Specify the top level output file name for your MegaCore function variation and click **Next** to display the MegaWizard interface **Parameter Settings** page. Use the MegaWizard interface to specify the required parameters for the MegaCore function variation. For information about the parameters available for each MegaCore function, refer to Chapter 3, Parameter Settings.

7. Click **Next** to complete the parameterization and display the **EDA** page. For example, Figure 2–3 shows the **EDA** page for the Chroma Resampler.

Figure 2–3. EDA Page for the Chroma Resampler



8. On the **EDA** page, turn on **Generate Simulation Model**.

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software.

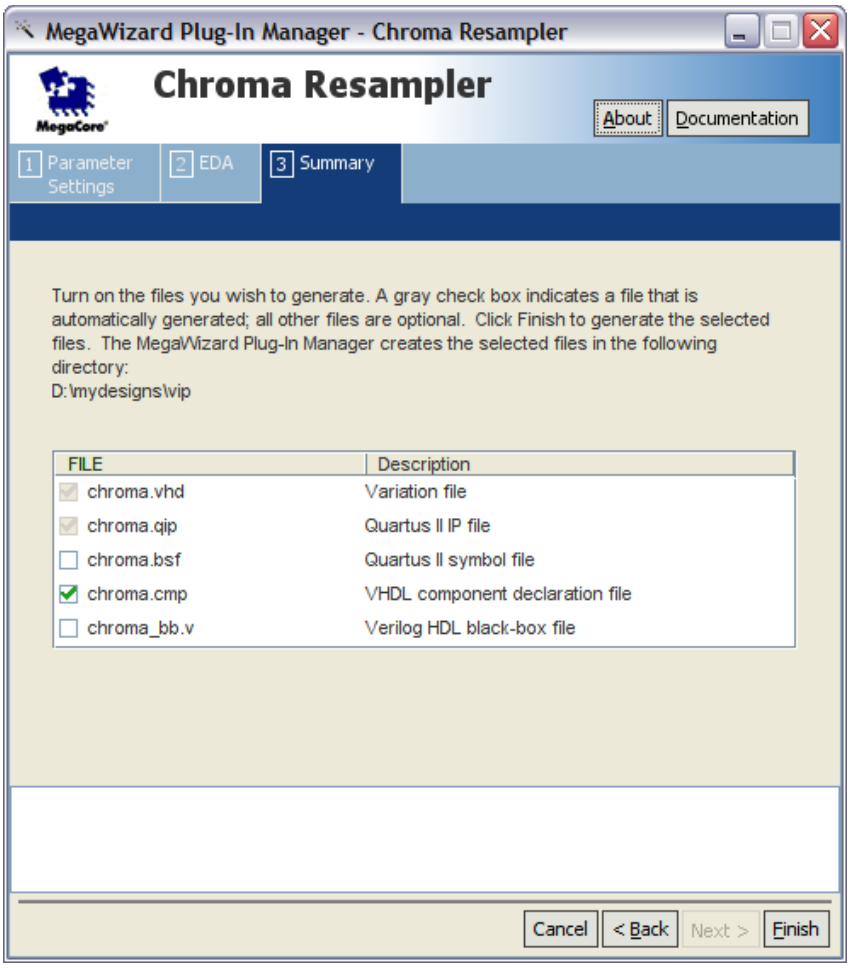
CAUTION

Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a non-functional design.

9. Some third-party synthesis tools can use a netlist that contains only the structure of the MegaCore function, but not detailed logic, to optimize performance of the design that contains the MegaCore function. If your synthesis tool supports this feature, turn on **Generate netlist**.

10. Click **Next** to display the **Summary** page. For example, Figure 2–4 shows the **Summary** page for the Chroma Resampler.

Figure 2–4. Summary Page for the Chroma Resampler



11. On the **Summary** tab, turn on the check boxes for the files you want to generate. A grey checkmark indicates a file that is automatically generated. All other files are optional.
12. Click **Finish** to generate the MegaCore function and supporting files. The generation phase may take several minutes to complete. The generation progress and status displays in a report window.
13. Click **Exit** to close the progress report window. Then click **Yes** on the **Quartus II IP Files** prompt to add the Quartus II IP file (**.qip**) file describing your custom MegaCore function variation to the current Quartus II project. For information about the **.qip** file, refer to Table 2–1 on page 2–6.

For more information about the MegaWizard Plug-In Manager flow, refer to the Quartus II Help.

Generated Files

Table 2–1 describes the generated files and other files that may be in your project directory.

The names and types of files vary depending on the variation name and HDL type you specify during parameterization For example, a different set of files are created based on whether you create your design in Verilog HDL or VHDL.

Table 2–1. Generated Files *(Note 1)*

File Name	Description
<*variation name*>.bsf	Quartus II block symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
<*variation name*>.cmp	A VHDL component declaration file for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.
<*variation name*>.qip	A single Quartus IP file is generated that contains all of the assignments and other information required to process your MegaCore function variation in the Quartus II compiler. In the SOPC Builder flow, this file is automatically included in your project. In the Megawizard plug-in Manager flow, you are prompted to add the **.qip** file to the current Quartus II project when you exit from the wizard. In SOPC Builder, a **.qip** file is generated for each MegaCore function and SOPC Builder component. Each of these **.qip** files are referenced by the system level **.qip** file and together include all the information required to process the system.
<*variation name*>.vhd, or **.v**	A VHDL or Verilog HDL file that defines the top-level description of the custom MegaCore function variation. Instantiate the entity defined by this file inside your design. Include this file when compiling your design in the Quartus II software.
<*variation name*>.vho or **.vo**	VHDL or Verilog HDL output files that defines an IP functional simulation model.
<*variation name*>_bb.v	A Verilog HDL black box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.
<*variation name*>_syn.v	A timing and resource estimation netlist for use in some third-party synthesis tools.

Note to Table 2–1:

(1) The <*variation name*> prefix is added automatically using the base output file name you specified in the MegaWizard interface.

For a full description of the signals supported on external ports for your MegaCore function variation, refer to "Signals" on page A–26.

Simulating the Design

You can simulate your design using the MegaWizard-generated VHDL or Verilog HDL IP functional simulation models (**.vo** or **.vho** generated files). Compile the file in your simulation environment and perform functional simulation of your custom MegaCore function variation.

For more information on IP functional simulation models, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook*.

Compiling the Design and Programming a Device

You can use the Quartus II software to compile your design, program a device, and verify your design in hardware.

For instructions on compiling and programming your design, and more information about the MegaWizard Plug-In Manager flow, refer to the Quartus II Help.

3. Parameter Settings

This chapter gives examples of how to parameterize a Video and Image Processing Suite MegaCore function.

The **Parameter Settings** page provides the same options whether the MegaWizard interface has been opened from the SOPC Builder flow or the MegaWizard Plug-In Manager flow.

The **EDA** and **Summary** tabs are not visible when you are using the SOPC Builder flow.

For information about opening the MegaWizard interface, refer to the "Design Flows" section in Chapter 2, Getting Started.

The parameters available depend on the MegaCore function you have selected. Some MegaCore functions provide multiple parameter setting pages.

The MegaWizard interface allows you to only select legal combinations of parameters, and warns you of any invalid configurations.

Color Space Converter

A typical application for a Color Space Converter is to convert Y'CbCr standard definition television images to R'G'B' for display on a computer monitor.

To parameterize the MegaCore function for this conversion, perform the following steps:

1. Set the parameters listed in Table 3–1 in the **General** tab of the **Parameter Settings** page (Figure 3–1 on page 3–2).

Table 3–1. Parameters for the Color Space Converter, General Tab

Parameter	Value
Color Plane Configuration	Three color planes in sequence.
Input Data Type	8 bits per pixel per color plane, unsigned data type, guard bands on with Max = 240 and Min = 16. *(Note 1)*
Output Data Type	8 bits per pixel per color plane, unsigned data type, guard bands off. *(Note 2)*
Result to output Data Type Conversion procedure	Leave these options with their default values until after you have set the parameters in the **Operands** tab.

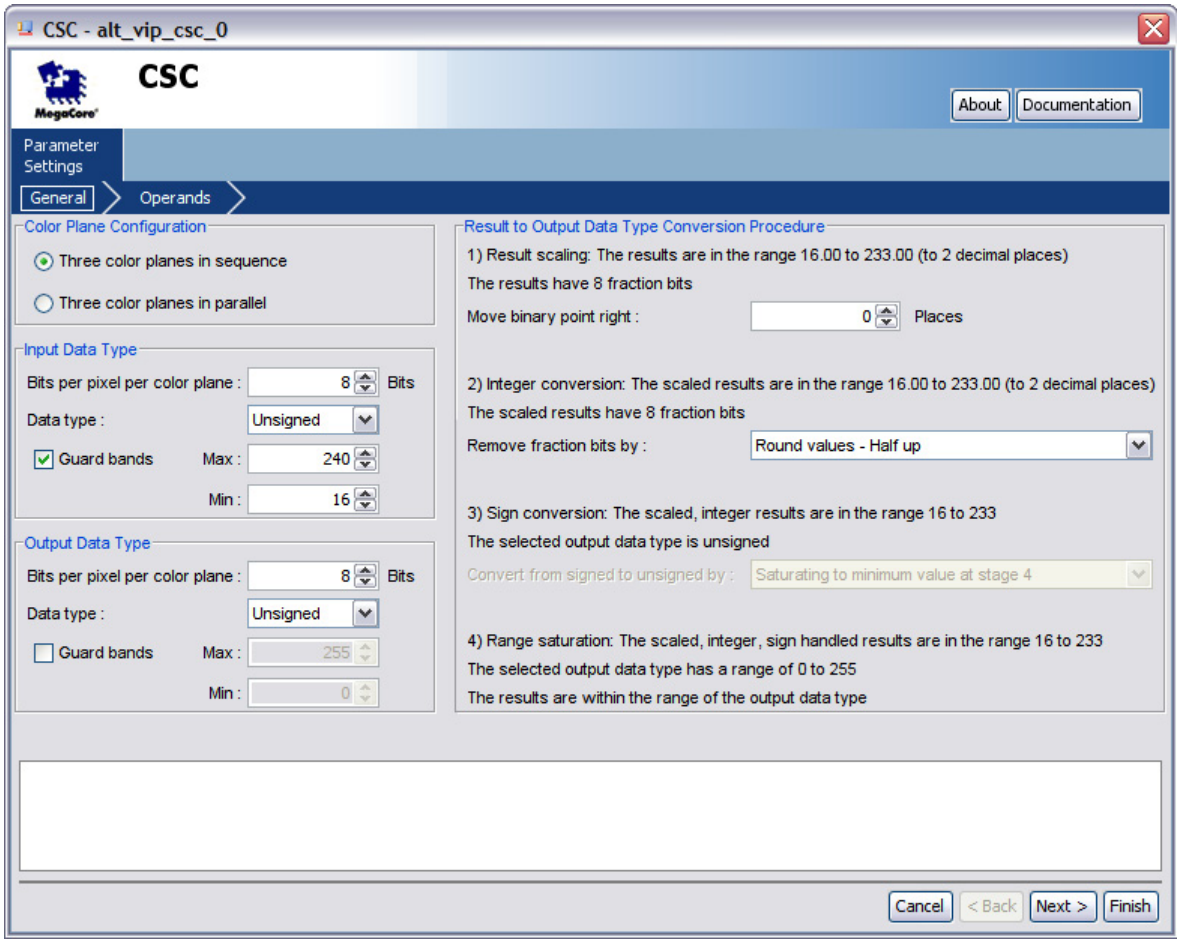
Notes to Table 3–1:

(1) These values mean that the MegaCore function never receives data in the guard bands, in this case between 241 to 255 and from 0 to 15.

(2) The output guard bands option is off because the full output range of 0 to 255 is required.

For more information about the options on the **General** tab of the Color Space Converter **Parameter Settings** page, refer to Table A–1 on page A–1.

Figure 3–1. General Parameter Settings for the Color Space Converter



2. Click **Next** to display the **Operands** tab of the **Parameter Settings** page (Figure 3–2 on page 3–3) and set the parameters listed in Table 3–2.

Table 3–2. Parameters for the Color Space Converter, Operands Tab

Parameter	Value
Run-time controlled	Off
Color Model Conversion	CbCrY': SDTV to Computer B'G'R' *(Note 1)*
Coefficients	Signed, integer bits = 2 *(Note 2)*
Summands	Signed, integer bits = 9 *(Note 2)*
Coefficient and summand fraction bits	7 *(Note 2)*

Notes to Table 3–2:

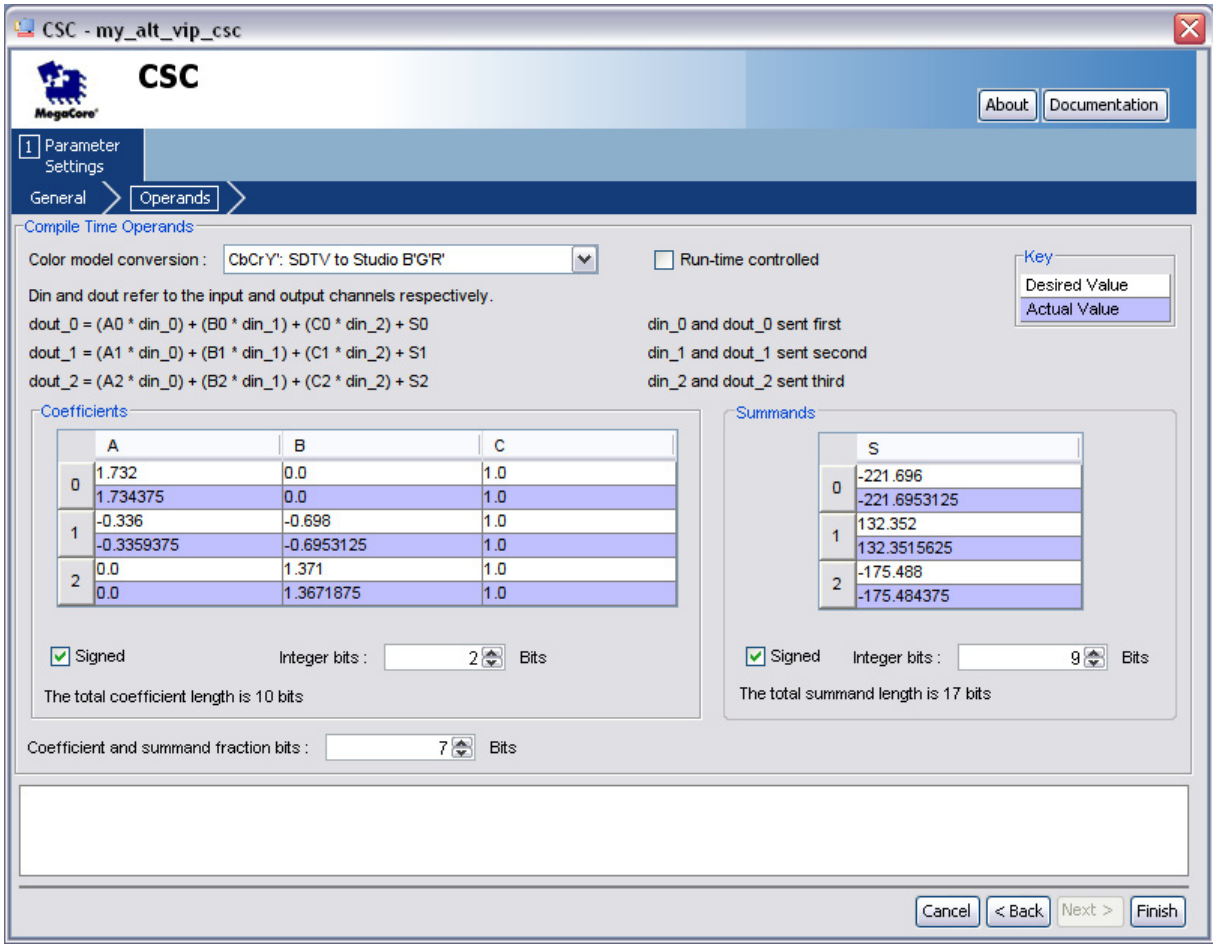
(1) Notice that the coefficient values are updated to preset values. If you edit these values, the color model conversion option changes to **Custom**.

(2) Notice how the actual coefficients and summands change. Signed coefficients allow negative values; increasing the integer bits increases the magnitude range; and increasing the fraction bits increases the precision.

(3) You may need to increase the number of bits available to avoid small numbers rounding to zero.

For more information about the options on the **Operands** tab of the Color Space Converter **Parameter Settings** page, refer to Table A–2 on page A–2.

Figure 3–2. Coefficients Parameter Settings for the Color Space Converter



3. Click **Back** to re-display the **General** tab of the **Parameter Settings** page.

   Notice that after changing the coefficients, the ranges shown under **Result to Output Data Type Conversion Procedure** have been updated.

4. Change the precision parameters on the **General** tab as listed in Table 3–3.

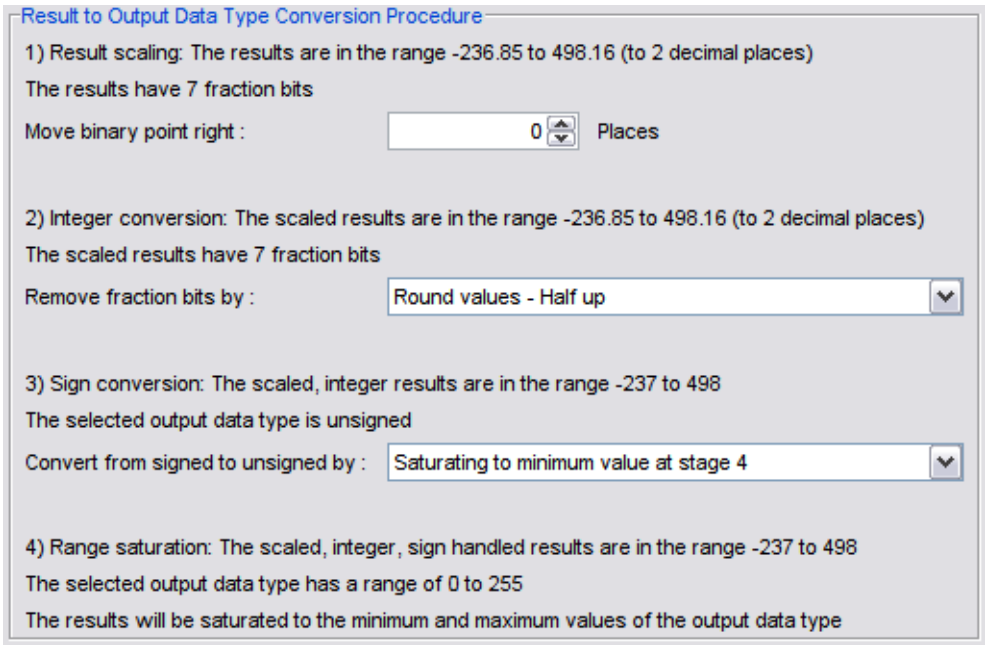
Table 3–3. Updated Parameters for the Color Space Converter, General Tab

Parameter	Value
Move binary point right	0
Remove fraction bits by	Round values - Half up
Convert from signed to unsigned by	Saturating to minimum value at stage 4

Notice that the scaled integer results no longer have a fractional part, the scaled, integer, sign handled results no longer include negative values but the result range is greater than the output data type range. The results must be constrained to this range. This is achieved by saturating to the minimum and maximum values of the output data range and is performed automatically.

Figure 3–3 shows the updated **Result to Output Data Type Conversion** section of the **General** tab in the **Parameter Settings** page.

Figure 3–3. Updated General Parameter Settings for the Color Space Converter



If a set of custom coefficients is required, you can type the values into the white cells in the tables on the **Operands** tab of the **Parameter Settings** page. Alternatively, you can paste custom coefficients into the table from a spreadsheet (such as Microsoft Excel). Blank lines must be left in your input data for the non-editable cells.

You can specify a higher precision output by increasing the output **Bits per pixel per color plane** and increasing the number of places specified by **Move binary point right**. Notice that the range of the results increases.

Chroma Resampler

To configure the Chroma Resampler to downsample a fully-sampled high-definition video stream to 4:2:2, follow these steps:

1. Set the parameters listed in Table 3–4 in the **Parameter Settings** page (Figure 3–4 on page 3–5).

Table 3–4. Parameters for the Chroma Resampler (Part 1 of 2)

Parameter	Value
Image width	1,920
Image height	1,080
Bits per pixel per color plane	8
Color plane configuration	Sequence

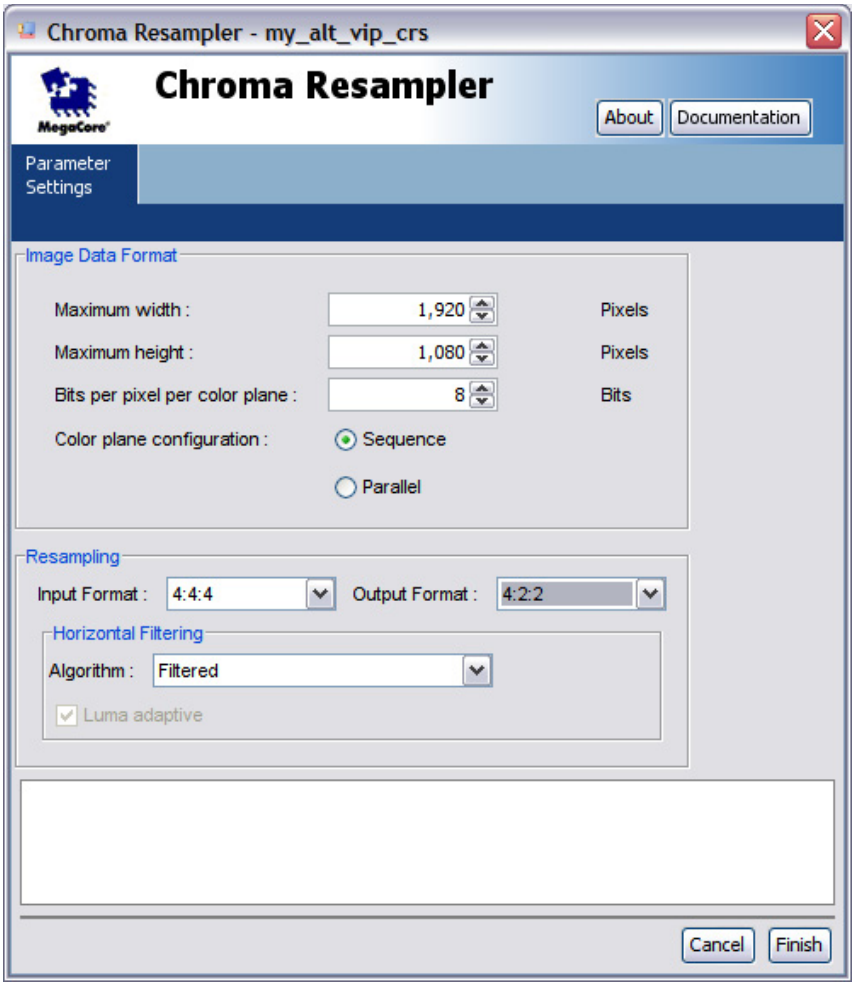
Table 3–4. Parameters for the Chroma Resampler (Part 2 of 2)

Parameter	Value
Input Format *(Note 1)*	4:4:4
Output Format *(Note 1)*	4:2:2
Algorithm	Filtered

Note to Table 3–4:

(1) The input and output formats must be different. A warning is issued when the same values are selected for both.

Figure 3–4. Parameter Settings for the Chroma Resampler



For more information about the options on the Chroma Resampler **Parameter Settings** page, refer to Table A–3 on page A–3.

Gamma Corrector

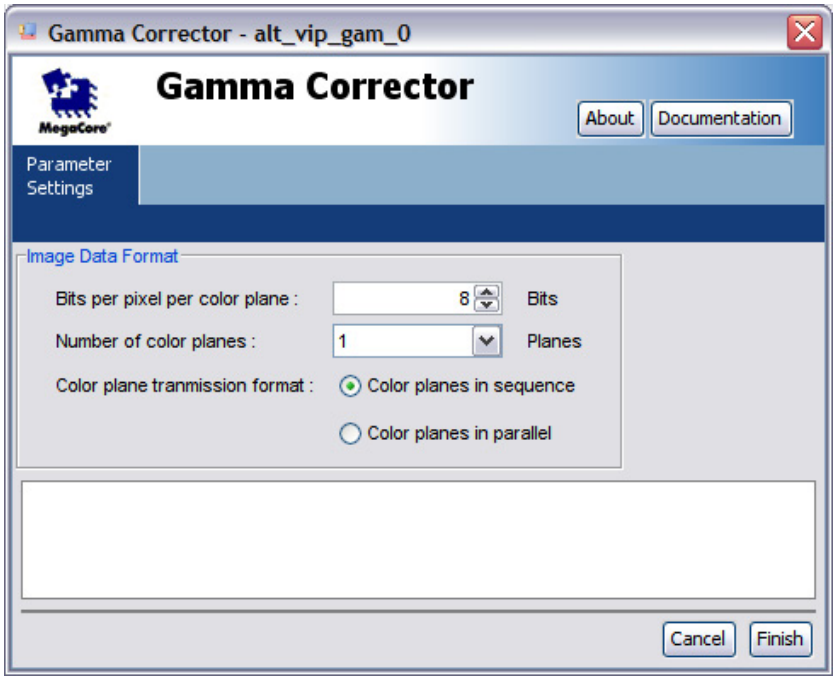
To configure your Gamma Corrector to correct a monochrome video stream of any supported resolution, set the image data format listed in Table 3–5 in the **Parameter Settings** page (Figure 3–5).

Table 3–5. Parameters for the Gamma Corrector

Parameter	Value
Bits per pixel per color plane	8
Number of color planes	1
Color planes in sequence	On

The actual gamma corrected intensity values are programmed at run time using the Avalon-MM slave interface.

Figure 3–5. Parameter Settings for the Gamma Corrector



For more information about the options on the Gamma Corrector **Parameter Settings** page, refer to Table A–4 on page A–3.

2D FIR Filter

A typical application of the 2D FIR Filter is to apply sharpening to a standard definition television picture converted to R'G'B'. To configure the 2D FIR to perform this sharpening operation, follow these steps:

1. Set the parameters listed in Table 3–6 in the **General** tab of the **Parameter Settings** page (Figure 3–6).

Table 3–6. Parameters for the **General** tab of the 2D FIR Filter Parameter Settings Page

Parameter	Value
Maximum image width	720 *(Note 1)*
Number of color planes in sequence	3
Input Data Type	8 bits per pixel per color plane, unsigned, guard bands off *(Note 2)*
Output Data Type	8 bits per pixel per color plane, unsigned, guard bands off *(Note 3)*

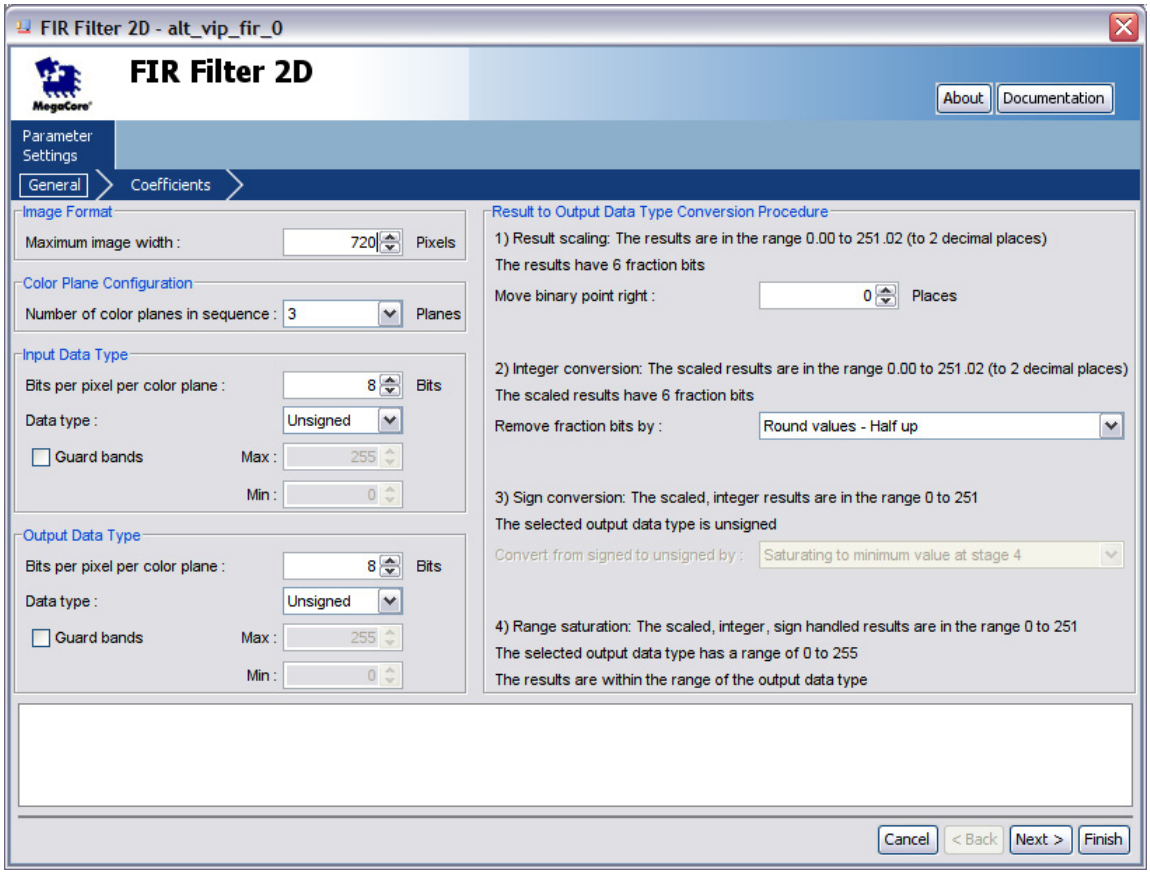
Note to Table 3–6:

(1) 720 pixels is the width of PAL video, a common standard-definition television format.

(2) When the input data guard bands is off, the entire 8-bit input range is used for video data. That is, the minimum value is 0 and the maximum value is 255.

(3) When the output data guard bands is off, the entire output range is allowed for video data.

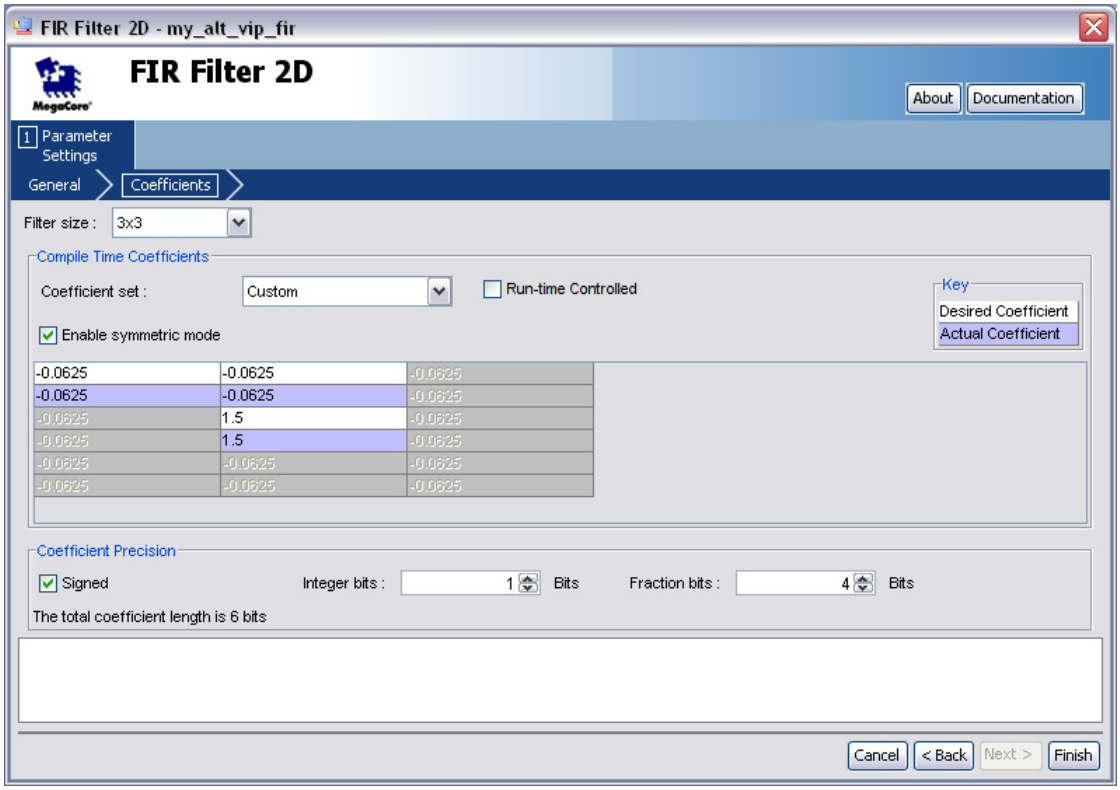
Figure 3–6. General Parameter Settings for the 2D FIR Filter



For more information about the options on the **General** tab of the 2D FIR Filter **Parameter Settings** page, refer to Table A–5 on page A–3.

2. Click **Next** to display the **Coefficients** tab of the **Parameter Settings** page (Figure 3–7).

Figure 3–7. Coefficients Parameter Settings for the 2D FIR Filter



3. Set the parameters listed in Table 3–7.

Table 3–7. Parameters for the Coefficients tab of the 2D FIR Filter Parameter Settings Page

Parameter	Value
Filter Size	3×3 *(Note 1)*
Run-time controlled	Off
Coefficient set	Simple Sharpening *(Note 1)*
Enable symmetric mode.	On

Note to Table 3–6:

(1) Notice that the size of the coefficient grid changes to match the filter size when this option is changed.

(2) Notice that the values in the coefficient grid change when you select a different coefficient set. The kernel is represented by a grid matrix where each coefficient is represented by two boxes. The white box contains the desired value, and the purple box shows the actual value for the current coefficient precision

(3) Notice that when symmetric mode is turned on, a limited number of matrix cells are editable and many of the values are automatically inferred. For example in Figure 3–7, values need only be edited in the three white cells. These values are automatically updated symmetrically in the remaining cells to complete the 3×3 matrix. A corresponding optimization reduces the number of multiplications that must be performed in the hardware.

4. Select the existing coefficient set by dragging the cursor across the matrix and use Ctrl+C to copy the entire matrix.
5. Change the selected coefficient set to **Custom**, select the top left cell and use Ctrl+V to paste the copied values into the matrix.
6. Edit the desired coefficients so that the sharpening is less strong as follows:
   a. Specify the central desired coefficient to be: `1.5`
   b. Specify the top left desired coefficient to be: `-0.0625`
   c. Specify the top central desired coefficient to be: `-0.0625`
7. Set the **Coefficient Precision** parameters listed in Table 3–8.

Table 3–8. Parameters for Coefficient Precision in the 2D FIR Filter General Tab

Parameter	Value
Signed	On
Integer bits	1
Fraction Bits	4

Notice how the actual coefficients change. Turning **Signed** on, allows negative values; increasing **Integer bits**, increases the magnitude range; and increasing **Fraction bits,** increases the precision. The actual coefficients should now closely match the desired coefficients. You may need to increase the number of bits available to avoid small numbers rounding to zero.

For more information about the options on the **Coefficients** tab of the 2D FIR Filter **Parameter Settings** page, refer to Table A–6 on page A–4.

8. Click **Back** to re-display the **General** tab of the **Parameter Settings** page.

   Notice that after changing the coefficients, the ranges shown under **Result to Output Data Type Conversion Procedure** have changed.
9. Change the precision parameters on the **General** tab as listed in Table 3–9.

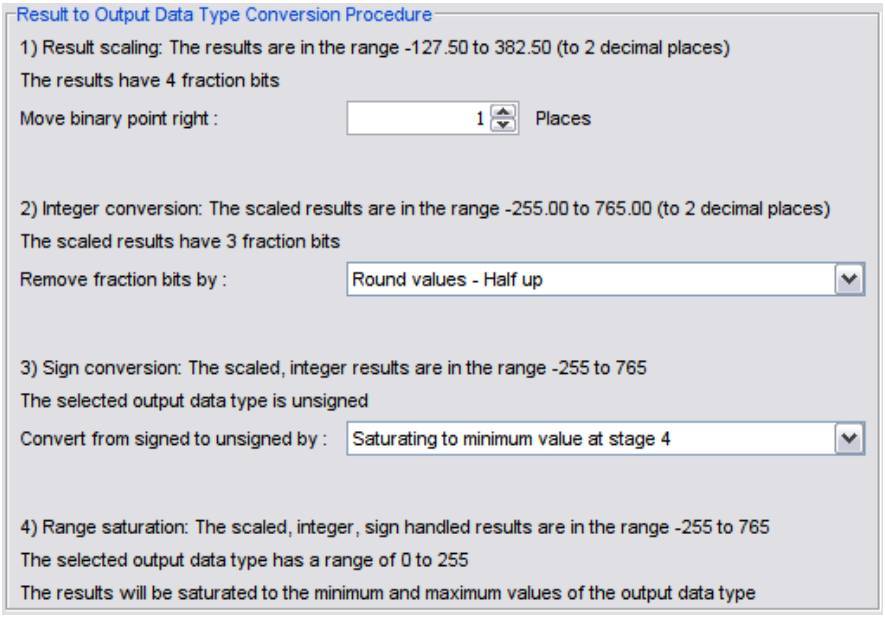
Table 3–9. Updated Parameters for the 2D FIR Filter General Tab

Parameter	Value
Move binary point right	1
Remove fraction bits by	Round values - Half up
Convert from signed to unsigned by	Saturating to minimum value at stage 4

Figure 3–8 on page 3–10 shows the updated **Result to Output Data Type Conversion Procedure** section of the **General** tab.

Notice that the scaled integer results no longer have a fractional part. Notice that the scaled, integer, sign handled results no longer include negative values but the result range is greater than the output data type range. The results are constrained to this range by saturating to the minimum and maximum values of the output data range (or output guard bands, when specified).

Figure 3–8. Updated General Parameter Settings for the 2D FIR Filter



You can specify a higher precision output by increasing the output **Bits per pixel per color plane** and increasing the number of places specified by **Move binary point right**. Notice that the range of the results increases. The result now has no fraction bits causing the **Remove fraction bits** control to be disabled.

2D Median Filter

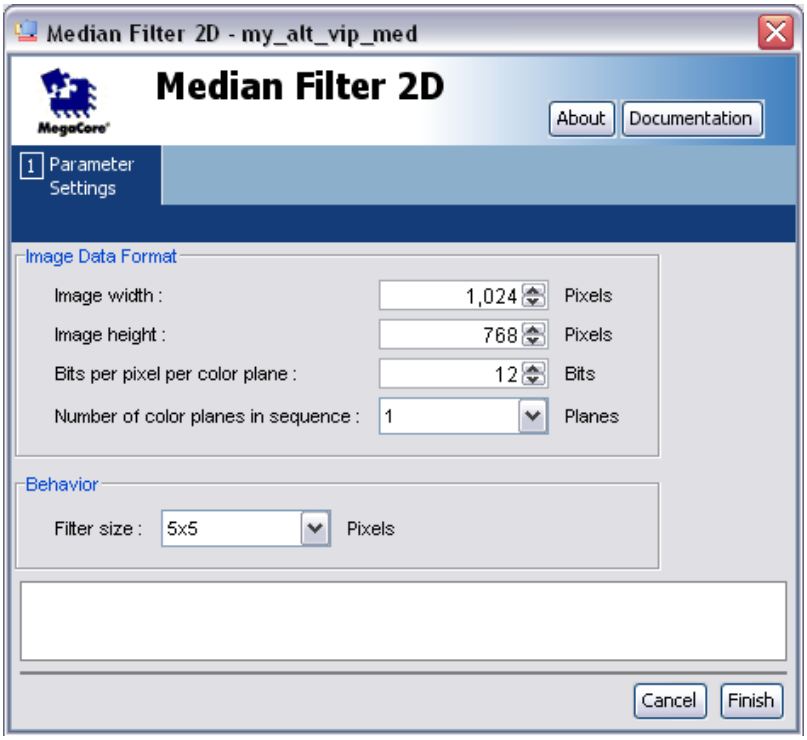
To configure your 2D Median Filter for 5×5 filtering of an example high resolution monochrome image format, set the parameters listed in Table 3–10 in the **Parameter Settings** page (Figure 3–9 on page 3–11).

Table 3–10. Parameters for the 2D Median Filter

Parameter	Value
Image width	1,024
Image height	768
Bits per pixel per color plane	12
Number of color planes in sequence	1
Filter size	5×5

For more information about the options on the 2D Median Filter **Parameter Settings** page, refer to Table A–7 on page A–5.

Figure 3–9. Parameter Settings for the 2D Median Filter



Alpha Blending Mixer

A typical application of the Alpha Blending Mixer is to layer an on-screen display and a picture-in-picture window over the top of a standard definition television picture.

To configure your Alpha Blending Mixer to perform this function, use the **Parameter Settings** page (Figure 3–10 on page 3–12) to specify the parameters listed in Table 3–11.

Table 3–11. Parameters for the Alpha Blending Mixer

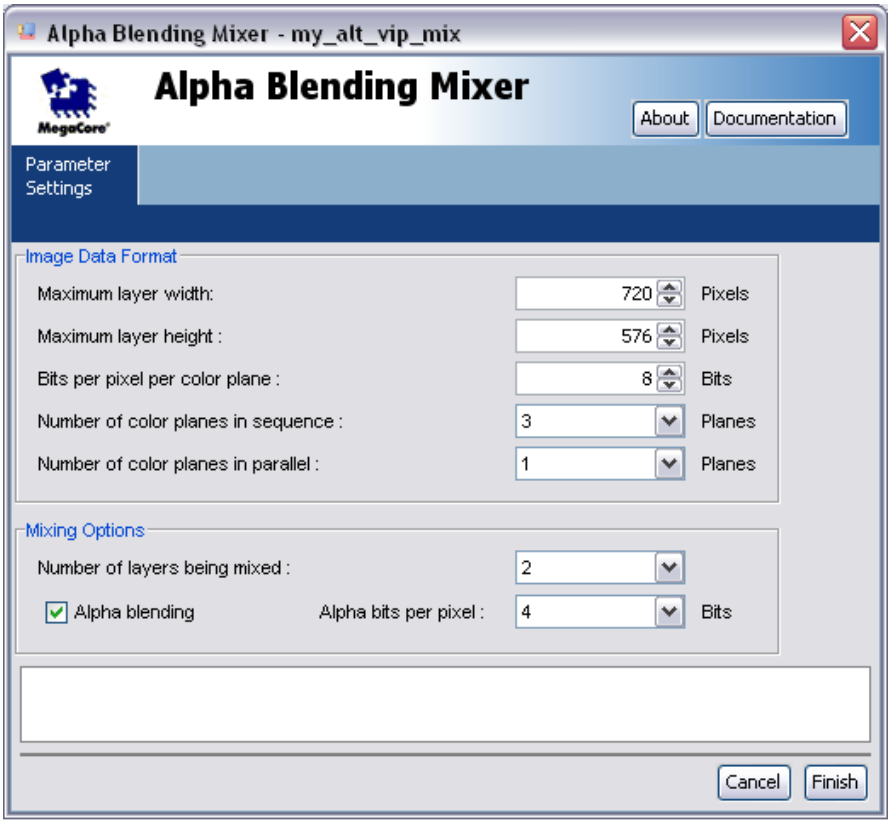
Parameter	Value
Maximum layer width	720 *(Note 1)*
Maximum layer height	576 *(Note 1)*
Bits per pixel per color plane	8
Number of color planes in sequence	3
Number of color planes in parallel	3
Number of layers being mixed	3
Alpha blending	On
Alpha bits per pixel	4 *(Note 2)*

Note to Table 3–11:

(1) This maximum layer width and height is the resolution of PAL video (a common standard television format). The resolutions used for each layer are specified at run time using the Avalon-MM interface.

(2) Notice that Alpha bits per pixel is available when you turn **Alpha blending** on. Setting this option to **4** allows the on-screen display to render semi-transparent graphics over the main display and the picture-in-picture window.

Figure 3–10. Parameter Settings for the Alpha Blending Mixer



For more information about the options on the Alpha Blending Mixer **Parameter Settings** page, refer to Table A–8 on page A–5.

Scaler

The Scaler MegaCore function resizes video streams and applies arbitrary coefficients. For example, to resize a video stream of resolution 640×480 to 1024×768 while at the same time applying a brightening effect, follow these steps:

1. Set the parameters listed in Table 3–12 in the **Resolution** tab of the **Parameter Settings** page (Figure 3–11 on page 3–13).

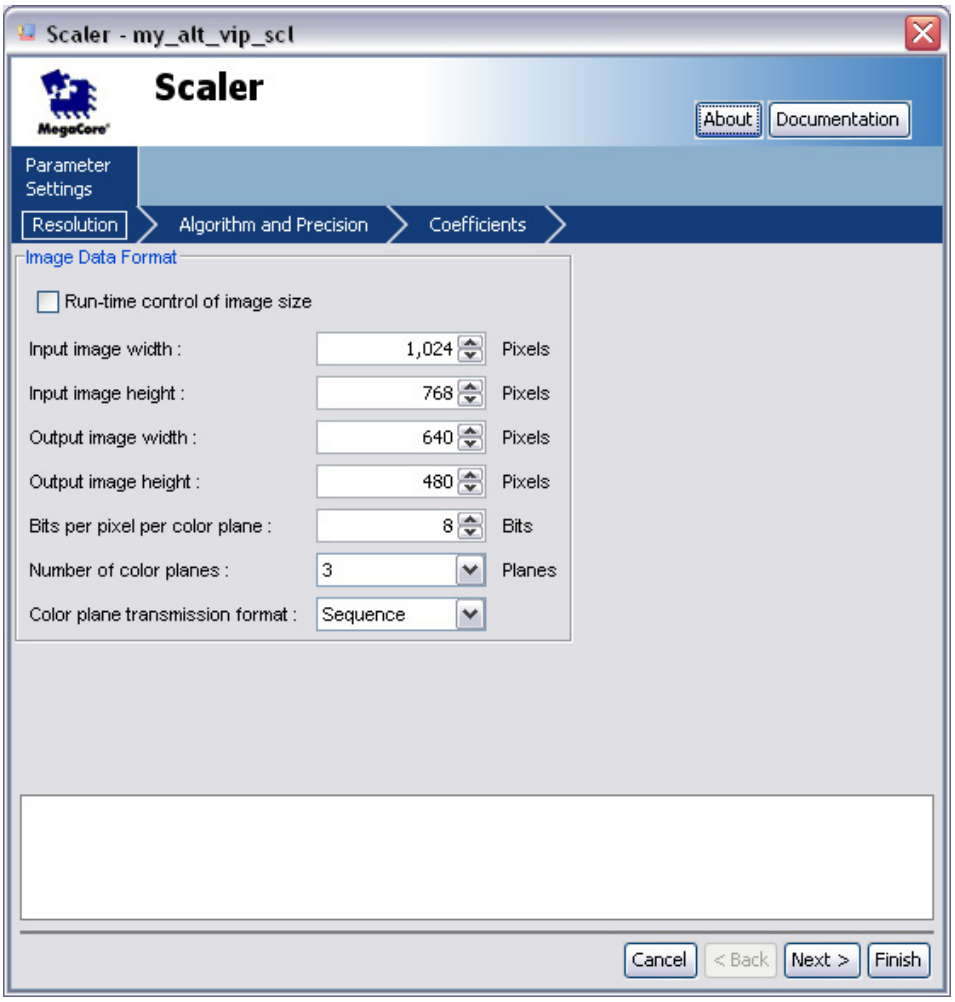
Table 3–12. Parameters for the Scaler Resolution Tab (Part 1 of 2)

Parameter	Value
Run-time control of image size	Off
Input image width	640
Input image height	480
Output image width	1,024
Output image height	768
Bits per pixel per color plane	8

Table 3–12. Parameters for the Scaler Resolution Tab (Part 2 of 2)

Parameter	Value
Number of color planes	3
Color plane transmission format	Sequence

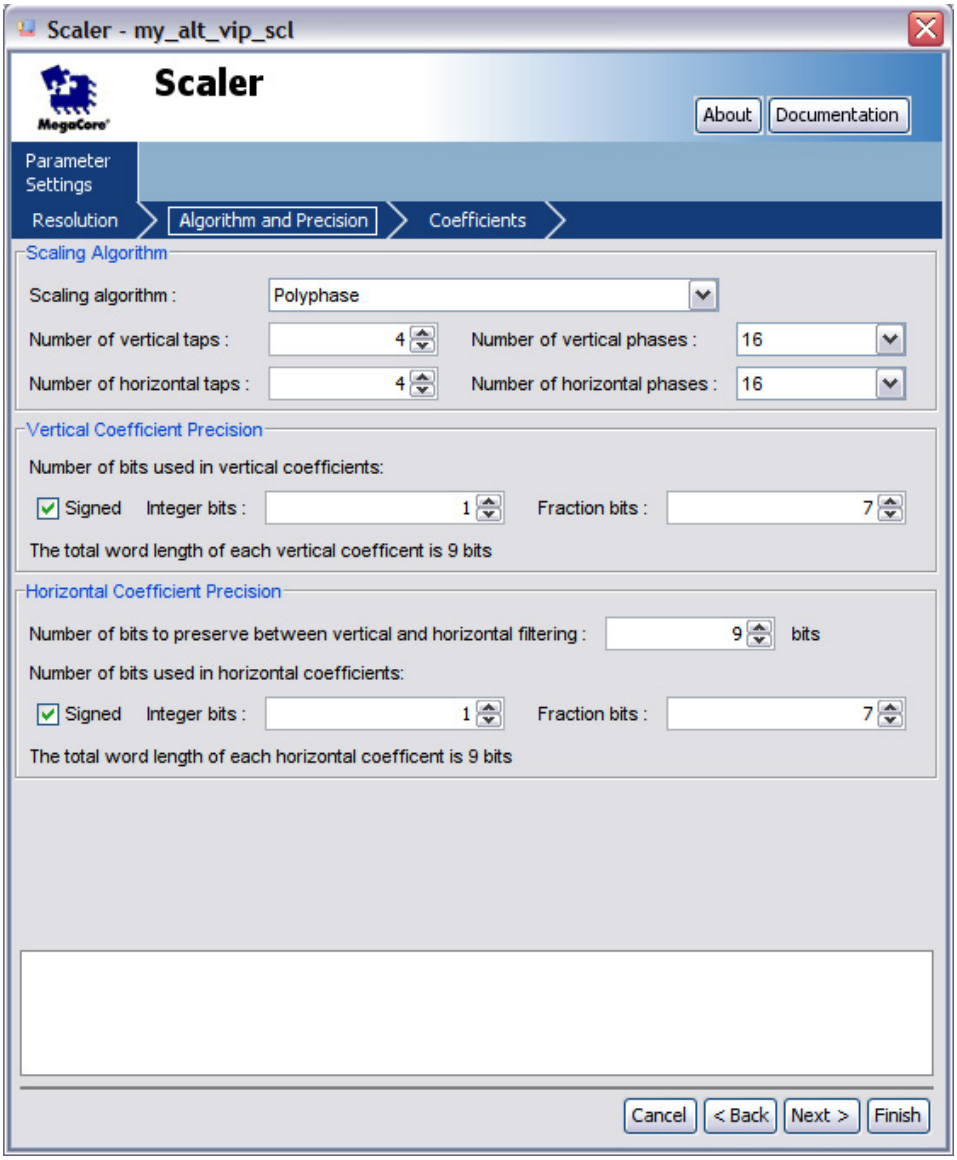
Figure 3–11. Resolution Parameter Settings for the Scaler



For more information about the options on the **Resolution** tab of the Scaler **Parameter Settings** page, refer to Table A–9 on page A–6.

2. Click **Next** to display the **Algorithm and Precision** tab of the **Parameter Settings** page (Figure 3–12 on page 3–14).

Figure 3–12. Algorithm and Precision Parameter Settings for the Scaler



3. Review the settings in the **Algorithm and Precision** page.

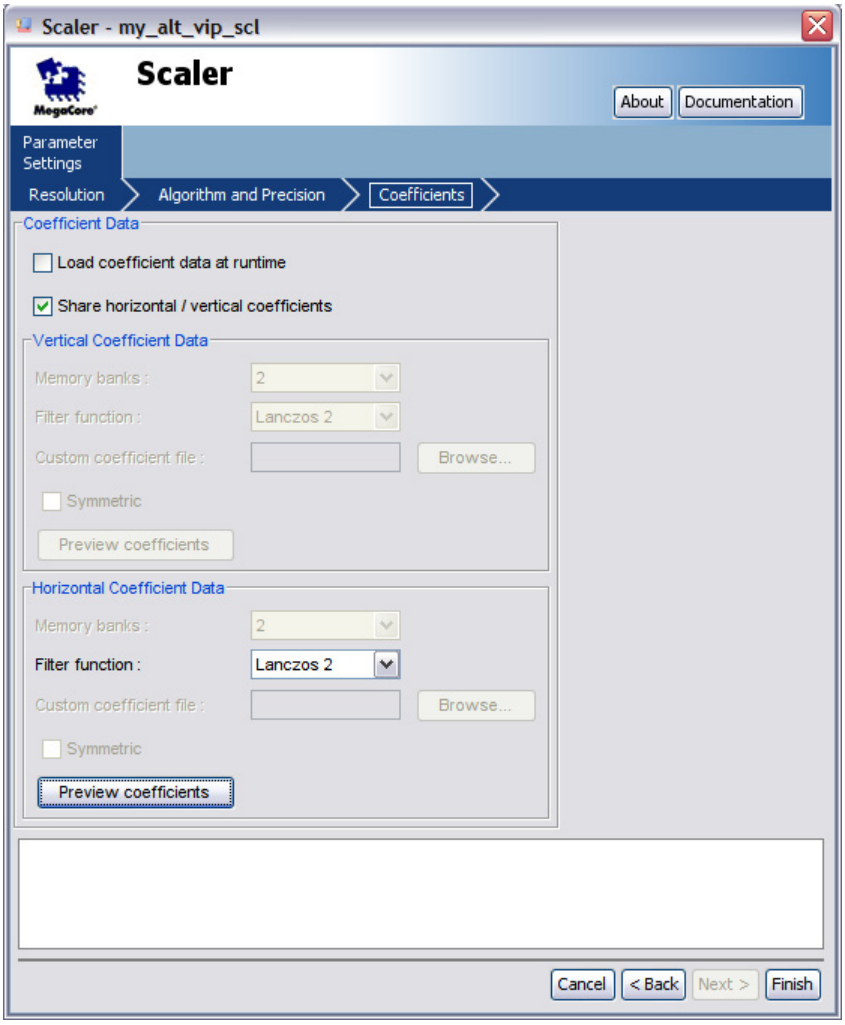
   The scaling algorithm should default to `Polyphase`. Leave all parameters on this page at their default values for this example.

   Notice that with signed coefficients, one integer bit, and seven fraction bits, the total word length of each coefficient is nine bits and that nine bits are preserved between vertical and horizontal filtering. This means that with four vertical and four horizontal taps, the scaler uses a total of eight 9×9 DSP blocks.

   For more information about the options on the **Algorithm and Precision** tab of the Scaler **Parameter Settings** page, refer to Table A–10 on page A–6.

4. Click **Next** to display the **Coefficients** tab of the **Parameter Settings** page (Figure 3–13).

Figure 3–13. Coefficients Parameter Settings for the Scaler



5. Set the parameters listed in Table 3–13.

Table 3–13. Parameters for the Scaler Coefficients Tab

Parameter	Value
Load coefficient data at runtime	Off
Share horizontal / vertical coefficients	On *(Note 1)*

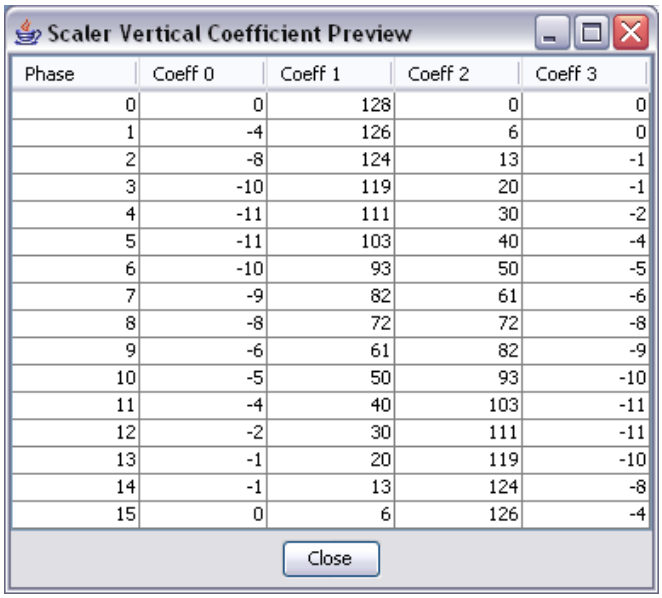
Note to Table 3–13:

(1) When this option is on, the coefficients are the same for vertical and horizontal data.

6. Click **Preview coefficients** under **Horizontal Coefficient Data** on the **Coefficients** page to view the quantized Lanczos 2 coefficients (Figure 3–14).
7. Use Shift+click and Ctrl+c to select all the Lanczos 2 coefficients and copy them to the clipboard.
8. Paste the coefficients into a suitable program such as Microsoft Excel or the MATLAB Array Editor and edit the data as follows:
   a. Delete the first column. This column indicates the phase and is not part of the required data.
   b. Multiply the remaining coefficient data by 1.1, convert it to integer type, and then export the data values to a comma-separated value (**.csv**) file.

   The 1.1 scaling factor increases the brightness of the resized image.

Figure 3–14. Scaler Default Lanczos 2 Coefficients



Phase	Coeff 0	Coeff 1	Coeff 2	Coeff 3
0	0	128	0	0
1	-4	126	6	0
2	-8	124	13	-1
3	-10	119	20	-1
4	-11	111	30	-2
5	-11	103	40	-4
6	-10	93	50	-5
7	-9	82	61	-6
8	-8	72	72	-8
9	-6	61	82	-9
10	-5	50	93	-10
11	-4	40	103	-11
12	-2	30	111	-11
13	-1	20	119	-10
14	-1	13	124	-8
15	0	6	126	-4

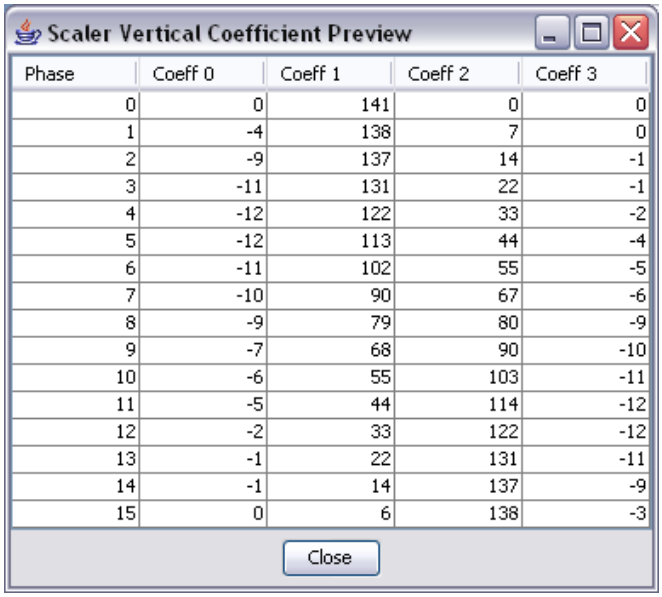
Each row of coefficients must sum to the same value. Refer to "Choosing and Loading Coefficients" on page 5–17.

9. Select a `Custom` **Filter function**, click **Browse**, and select the **.csv** file that you created in step 8.

These custom coefficient are applied to both the vertical and horizontal data because you selected **Share horizontal / vertical coefficients** in step 5.

10. Click **Preview coefficients** to confirm that the data has been read correctly (Figure 3–15 on page 3–17).

Figure 3–15. Scaler Custom Coefficients



Phase	Coeff 0	Coeff 1	Coeff 2	Coeff 3
0	0	141	0	0
1	-4	138	7	0
2	-9	137	14	-1
3	-11	131	22	-1
4	-12	122	33	-2
5	-12	113	44	-4
6	-11	102	55	-5
7	-10	90	67	-6
8	-9	79	80	-9
9	-7	68	90	-10
10	-6	55	103	-11
11	-5	44	114	-12
12	-2	33	122	-12
13	-1	22	131	-11
14	-1	14	137	-9
15	0	6	138	-3

For more information about the options on the **Coefficients** tab of the Scaler **Parameter Settings** page, refer to Table A–11 on page A–7.

Clipper

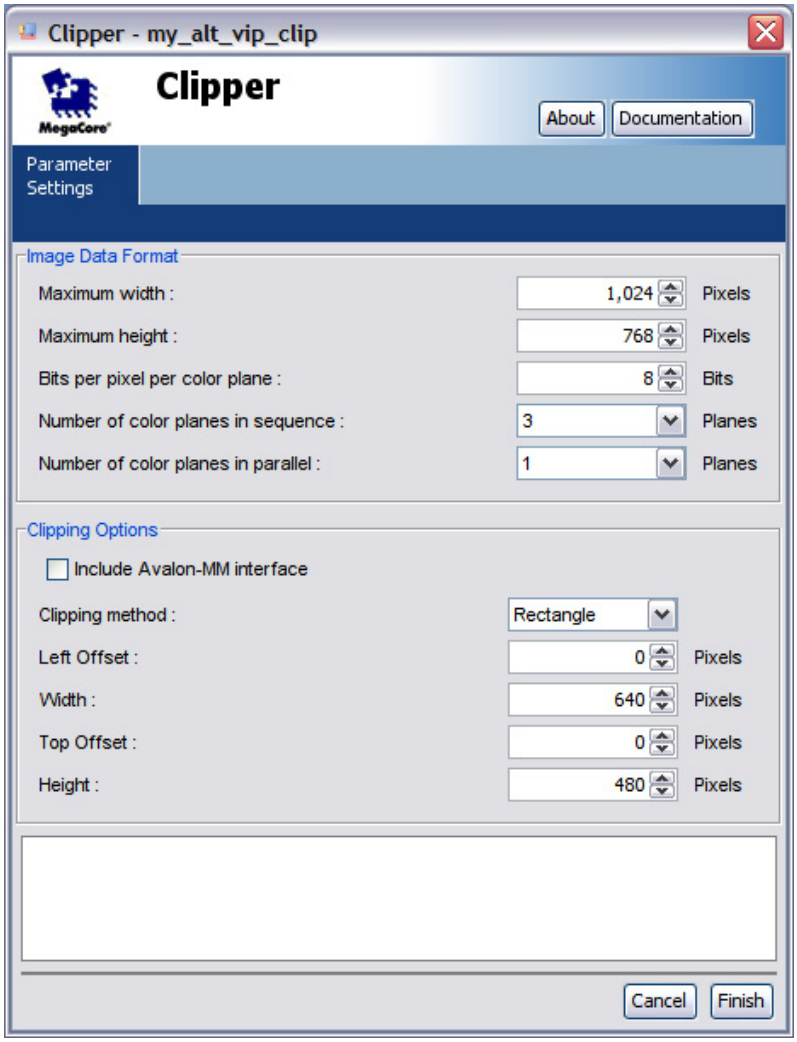
The Clipper MegaCore function can be parameterized to crop the upper-left 640×480 pixels of a 1024×768 video stream. If the resolution changes (to a value in the range 640×480 to 1024×768), the Clipper can adapt to the change and keep sending the same cropped rectangle without any further reconfiguration. To set up such a Clipper, follow these steps:

1. Set the parameters listed in Table 3–14 the **Parameter Settings** page (Figure 3–16 on page 3–18).

Table 3–14. Parameters for the Clipper

Parameter	Value
Maximum width	1,024
Maximum height	768
Bits per pixel per color plane	8
Number of color planes in sequence	3
Number of color planes in parallel	1
Include Avalon-MM interface: Off	Off
Clipping method	Rectangle
Left Offset	0
Width	640
Top Offset	0
Height	480

Figure 3–16. Parameter Settings for the Clipper



For more information about the options on the Clipper **Parameter Settings** page, refer to Table A–12 on page A–8.

Deinterlacer

To configure your Deinterlacer function to convert NTSC video input to progressive output at 30 frames per second using the weave deinterlacing algorithm, follow these steps:

1. Set the parameters listed in Table 3–15 in the **Parameter Settings** page (Figure 3–17 on page 3–20).

Table 3–15. Parameters for the Deinterlacer (Part 1 of 2)

Parameter	Value
Maximum image width	720
Maximum image height	486

Table 3–15. Parameters for the Deinterlacer (Part 2 of 2)

Parameter	Value
Bits per pixel per color plane	8
Number of color planes in sequence	2
Number of color planes in parallel	1
Default initial field	F0
Deinterlacing method:	Weave *(Note 1)*
Frame buffering mode	Double buffering *(Note 1)*, *(Note 3)*, *(Note 4)*, *(Note 5)*
Output frame rate	As input frame rate (F1 synchronized)
Passthrough mode (propagate progressive frames unchanged)	Off
Motion bleed	Off *(Note 2)*
Runtime control of the motion-adaptive blending	Off *(Note 2)*, *(Note 6)*
Runtime control for locked frame rate conversion	Off *(Note 4)*, *(Note 6)*
Use separate clocks for the Avalon-MM master interfaces	Off
Avalon-MM master ports width	128 *(Note 3)*
Read-only master(s) interface FIFO depth	64
Read-only master(s) interface burst target	32
Write-only master(s) interface FIFO depth	64
Write-only master(s) interface burst target	32
Base address of frame buffers	0x00001000 *(Note 3)*
Number of packets buffered per field	1 *(Note 5)*
Maximum packet length	10 *(Note 5)*

Notes to Table 3–15:

(1) Either double or triple-buffering mode must be selected before you can select the weave or motion-adaptive deinterlacing methods.

(2) These options are available only when you select Motion Adaptive as the deinterlacing method.

(3) The options to specify the Avalon-MM master ports width and the base address for the frame buffers are available only when you select double or triple-buffering.

(4) The option to synchronize input and output frame rates is only available when double-buffering mode is selected.

(5) The options to control the buffering of non-image data packets are available when you select double or triple-buffering.

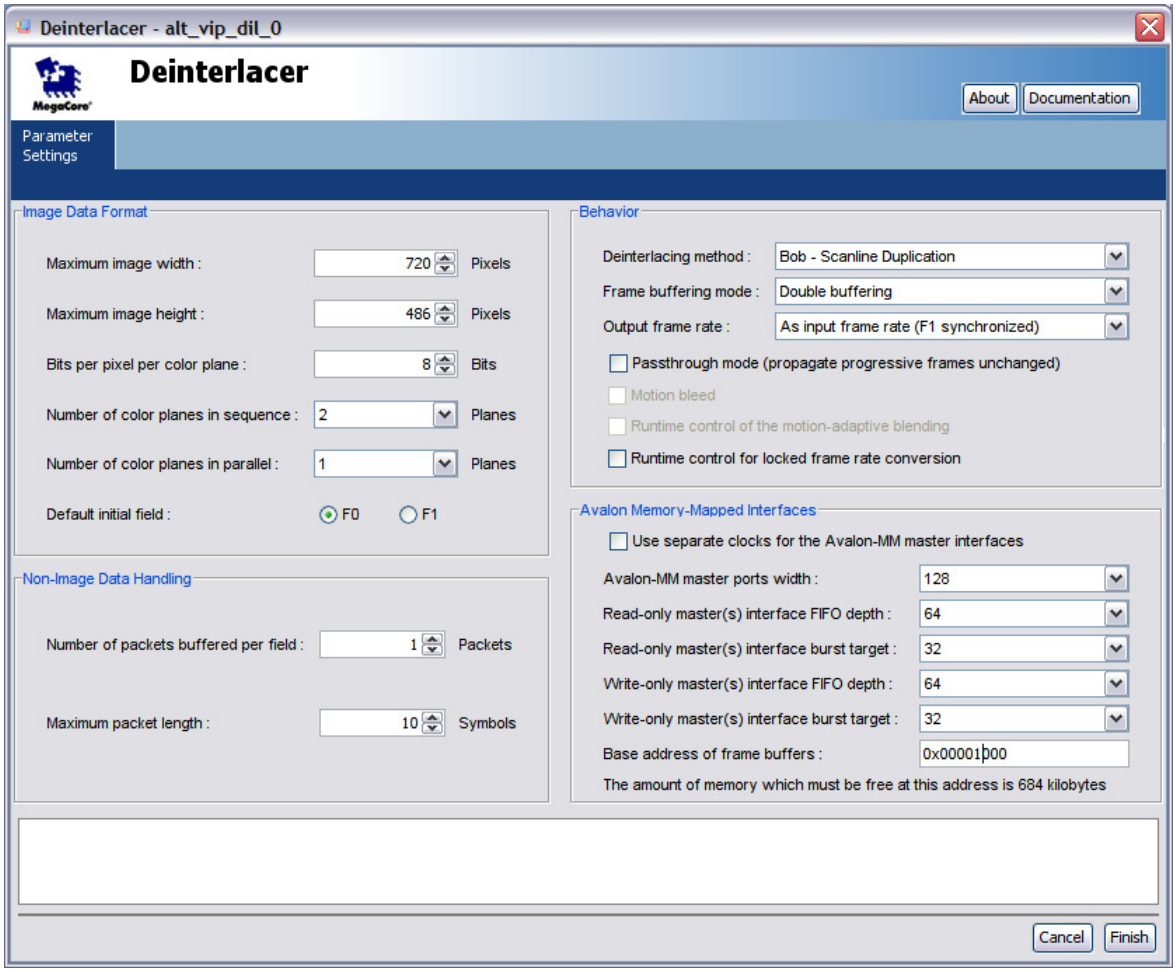
(6) You cannot enable both run-time control interfaces at the same time.

The memory required at the specified hexadecimal base address is displayed under the options for **Avalon Memory-Mapped Interfaces**. NTSC video transmits 60 interlaced fields per second, and therefore 30 frames per second. Selecting the output frame rate to be the same as the input frame rate ensures that the output will be at 30 frames per second.

The weave and motion-adaptive algorithms work by stitching/blending together lines from two fields, a current field and the field preceding the current field. An output frame rate of F1 synchronized means that each current field is an F1 field. The weave algorithm stitches together F1 fields with the F0 fields that precede rather than follow them. Similarly, the motion-adaptive algorithm uses the F0 fields that precede the current F1 field to build an output frame.

 A F0 field contains the top line, any other field is a F1 field.

Figure 3–17. Parameter Settings for the Deinterlacer



For more information about the options on the Deinterlacer **Parameter Settings** page., refer to Table A–13 on page A–9.

Frame Buffer

To parameterize your Frame Buffer function to allow for triple-buffering of a 480×720 R′G′B′ video stream transmitted in parallel:

1. Set the parameters listed in Table 3–16 in the **Parameter Settings** page (Figure 3–18 on page 3–21).

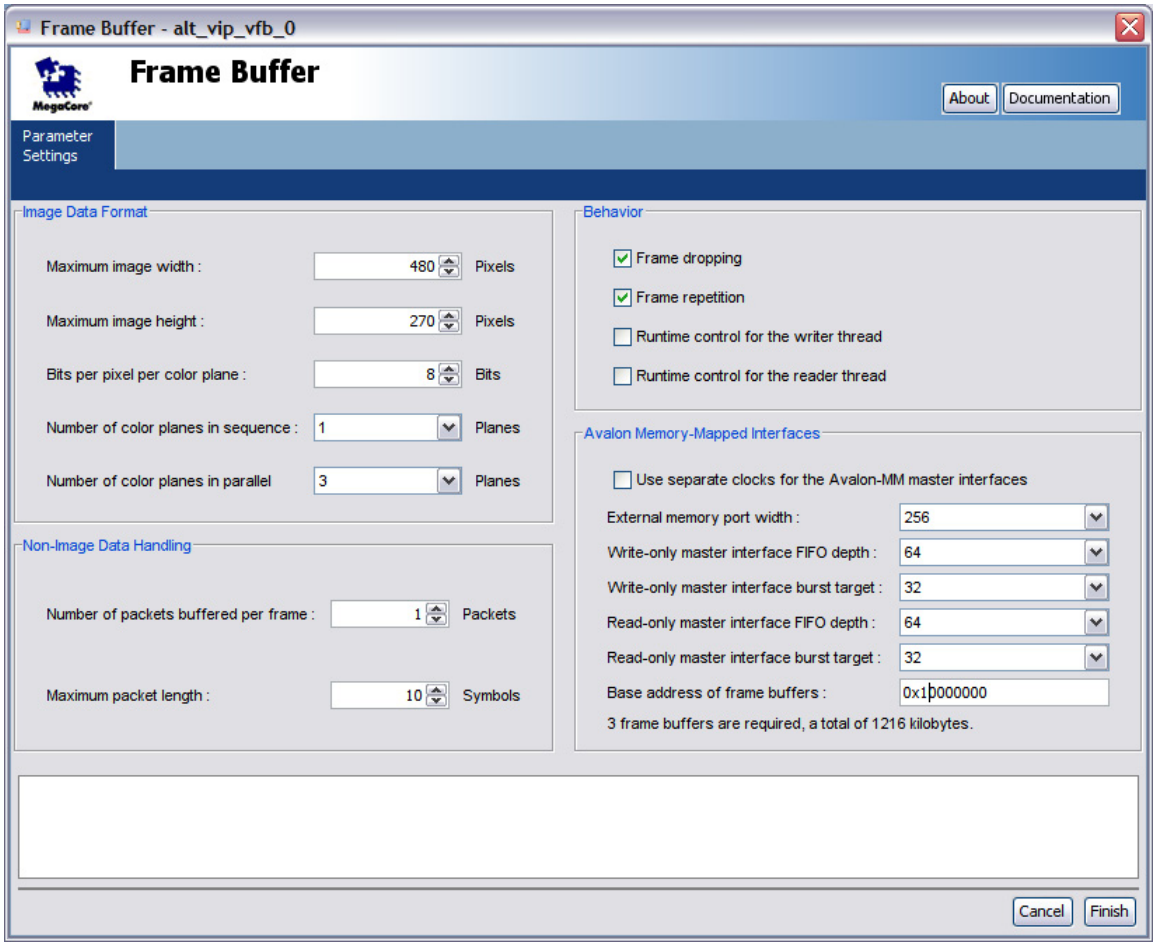
Table 3–16. Parameters for the Frame Buffer (Part 1 of 2)

Parameter	Value
Maximum image width	480
Maximum image height	270
Bits per pixel per color plane	8
Number of color planes in sequence	1

Table 3–16. Parameters for the Frame Buffer (Part 2 of 2)

Parameter	Value
Number of color planes in parallel	3
Frame dropping: On	On
Frame repetition	On
Runtime control for the writer thread	Off
Runtime control for the reader thread	Off
Use separate clocks for the Avalon-MM master interfaces	Off
External memory port width	256
Write-only master interface FIFO depth	64
Write-only master interface burst target	32
Read-only master interface FIFO depth	64
Read-only master interface burst target	32
Base address of frame buffers	0x10000000
Number of packets buffered per frame	1
Maximum packet length	10

Figure 3–18. Parameter Settings for the Frame Buffer



The number of frame buffers required and the size in kilobytes is displayed under the options for **Avalon Memory-Mapped Interfaces**.

For more information about the options on the Frame Buffer **Parameter Settings** page, refer to Table A–14 on page A–10.

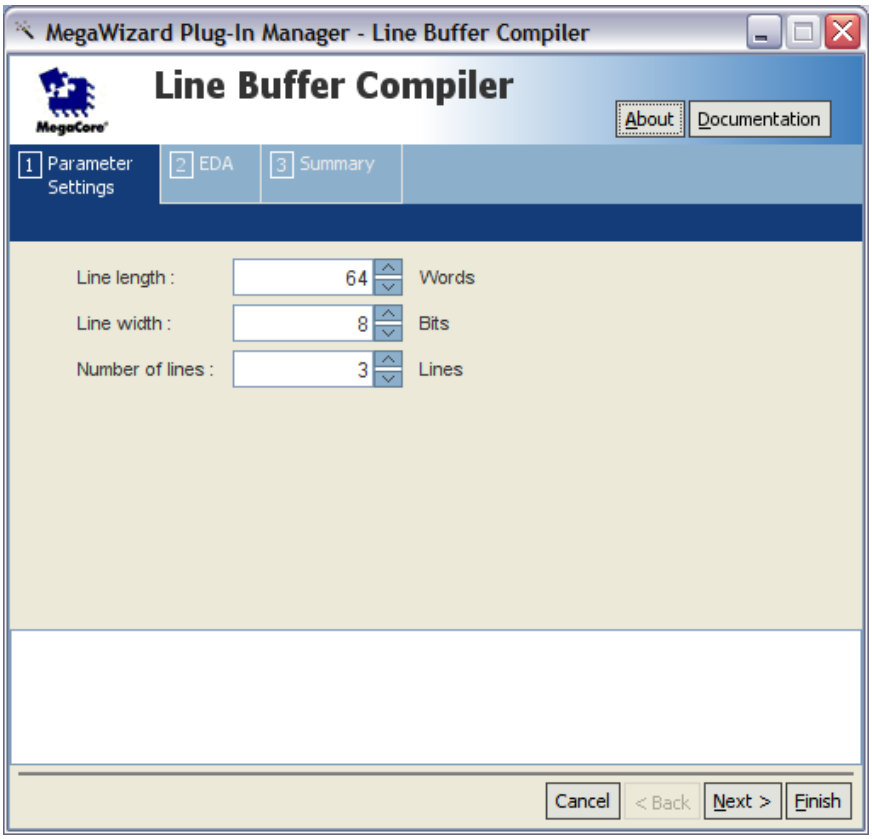
Line Buffer Compiler

To parameterize your Line Buffer Compiler function for a set of four line buffers each capable of holding 320 24-bit words, specify the parameters listed in Table 3–17 in the **Parameter Settings** page (Figure 3–19).

Table 3–17. Parameters for the Line Buffer Compiler

Parameter	Value
Line length	320
Line width	24
Number of lines	4

Figure 3–19. Parameter Settings for the Line Buffer Compiler



The Line Buffer Compiler is not available in the SOPC Builder design flow.

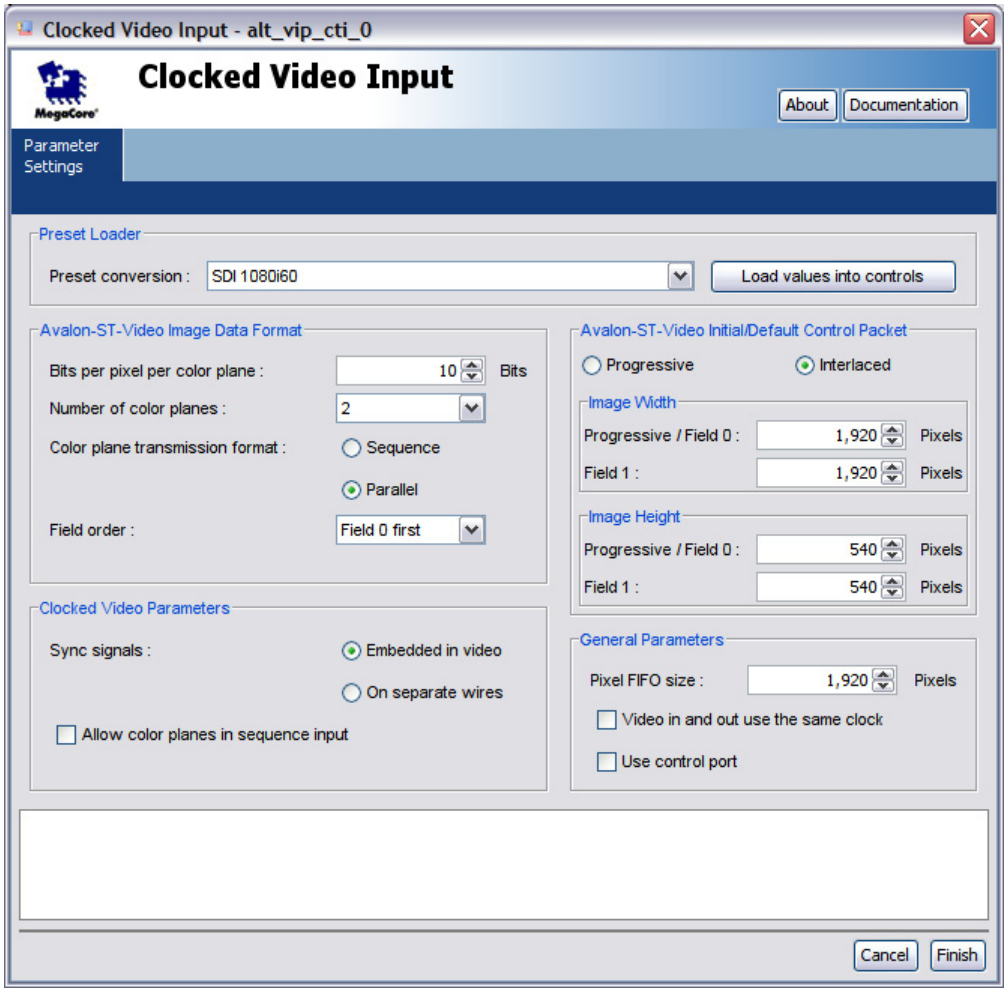
For more information about the options on the Line Buffer Compiler **Parameter Settings** page, refer to Table A–15 on page A–11.

Clocked Video Input

To parameterize your Clocked Video Input function for converting a high definition (HD) 1080i60 video stream (such as BT656 video from a serial digital interface, perform the following steps:

1. Select **SDI 1080i60** from the list of preset conversions in the **Parameter Settings** page (Figure 3–20) and click **Load values into controls** to initialize the other parameters in the MegaWizard interface.

Figure 3–20. Parameter Settings for the Clocked Video Input



2. Set the additional parameters listed in Table 3–18 in the **Parameter Settings** page.

Table 3–18. Parameters for the Clocked Video Input (Part 1 of 2)

Parameter	Value
Bits per pixel per color plane	10
Number of color planes	2
Color plane transmission format	Parallel
Field order	Field 0 first

Table 3–18. Parameters for the Clocked Video Input (Part 2 of 2)

Parameter	Value
Avalon-ST Video Initial/Default Control Packet	Interlaced
Image Width, Progressive/Field 0	1,920
Image Width, Field 1	1,920
Image Height, Progressive/Field 0	540
Image Height, Field 1	540
Sync Signals	Embedded in video
Allow color planes in sequence input: Off	Off
Pixel FIFO size	1,920
Video in and out use the same clock	Off
Use control port	Off

For more information about the options on the Clocked Video Input **Parameter Settings** page, refer to Table A–16 on page A–12.

Clocked Video Output

To parameterize your Clocked Video Output function for creating a DVI 1080p video stream:

1. Select `DVI 1080p60` from the list of preset conversions in the **Parameter Settings** page (Figure 3–21 on page 3–25) and click **Load values into controls** to initialize the other parameters in the MegaWizard interface.
2. Set the additional parameters listed in Table 3–19 in the **Parameter Settings** page.

Table 3–19. Parameters for the Clocked Video Output (Part 1 of 2)

Parameter	Value
Image width / Active pixels: 1,920	1,920
Image height / Active lines	1,080
Bits per pixel per color plane	8
Number of color planes: 3	3
Color plane transmission format	Parallel
Allow output of color planes in sequence	Off
Interlaced video: Off	Off
Sync Signals	On separate wires
Active picture line	0
Horizontal sync	60
Horizontal front porch	20
Horizontal back porch	192
Vertical sync	5
Vertical front porch	4
Vertical back porch	36
Pixel FIFO size	1,920

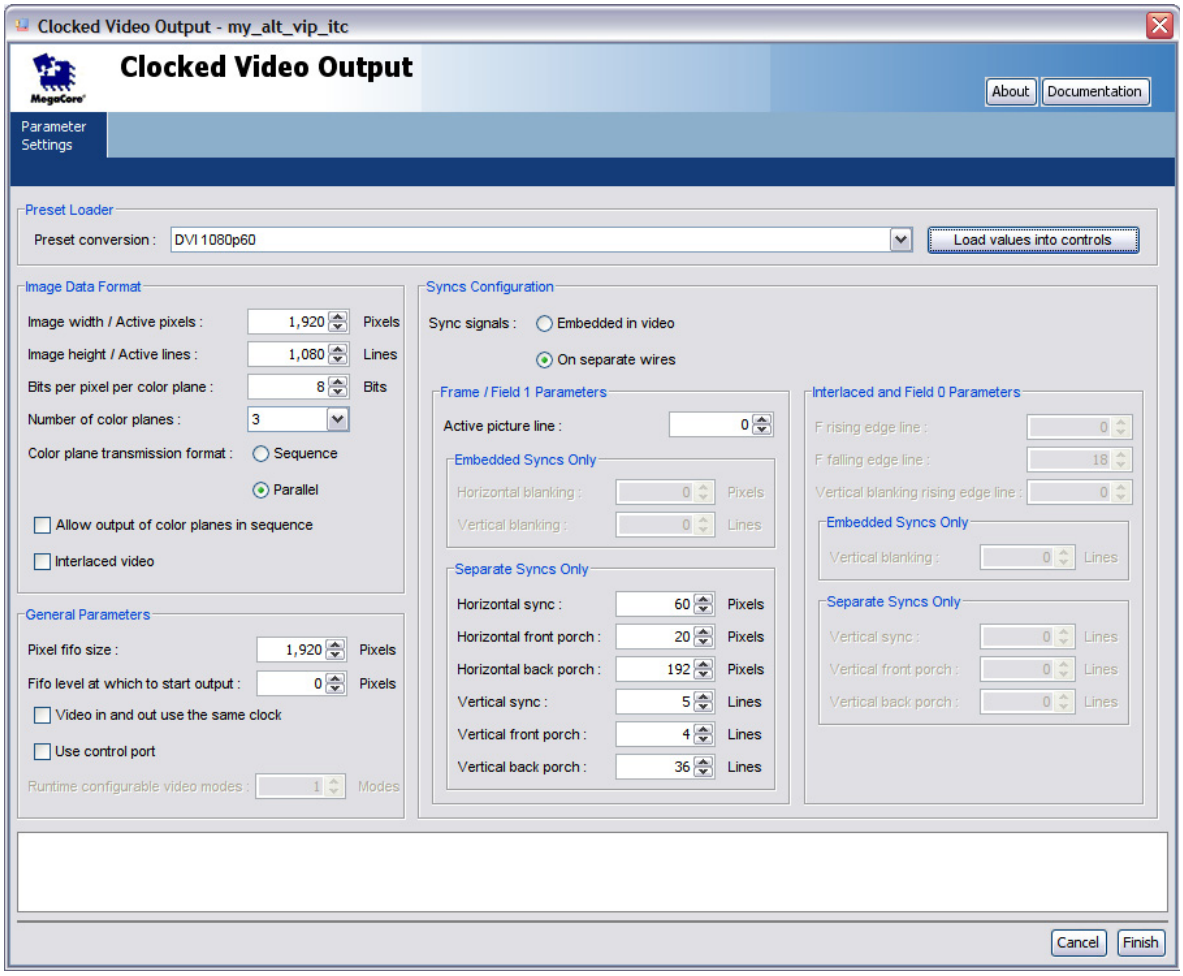
Table 3–19. Parameters for the Clocked Video Output (Part 2 of 2)

Parameter	Value
FIFO Level at which to start output	0
Video in and out use the same clock	Off
Use control port	Off *(Note 1)*
Runtime configurable video modes	1 *(Note 1)*

Note to Table 3–19:

(1) The option to specify the number of run-time configurable video modes is available only when **Use Control Port** is on.

Figure 3–21. Parameter Settings for the Clocked Video Output



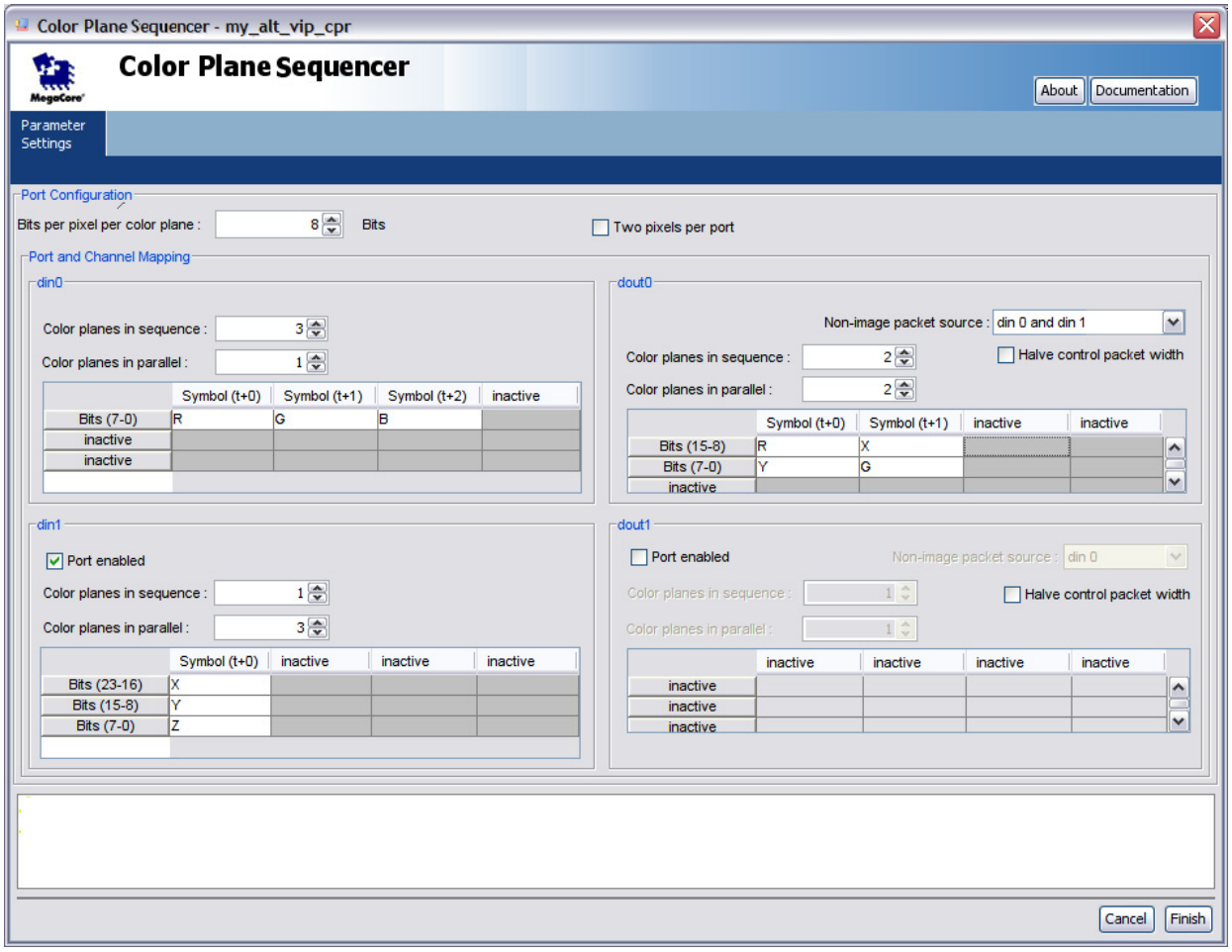
For more information about the options on the Clocked Video Output **Parameter Settings** page, refer to Table A–17 on page A–13.

Color Plane Sequencer

This section includes two examples for parameterizing the Color Plane Sequencer MegaCore function.

The first example uses a Color Plane Sequencer to combine two Avalon-ST Video streams (an R′G′B′ color pattern as three color planes in sequence and another pattern X′Y′Z′ as three color planes in parallel) as a single stream. The Z′ and B′ color planes are discarded and non-video packets are accepted from either input port.

Figure 3–22. Parameter Settings for the Color Plane Sequencer (Combining Streams Example)



For this example, set the parameters listed in Table 3–20 in the **Parameter Settings** page (Figure 3–22).

Table 3–20. Parameters for the Color Plane Sequencer (Combining Streams Example) (Part 1 of 2)

	Parameter	Value
	Bits per pixel per color plane	8
	Two pixels per port	Off
din0	Color planes in parallel	1
	Color planes in sequence	3

Table 3–20. Parameters for the Color Plane Sequencer (Combining Streams Example) (Part 2 of 2)

	Parameter	Value
	Bits (7–0), Symbol (t+0)	R
	Bits (7–0), Symbol (t+1)	G
	Bits (7–0), Symbol (t+2)	B
din1	Port enabled	On
	Color planes in parallel	3
	Color planes in sequence	1
	Bits (23–16), Symbol (t+0)	X
	Bits (15–8), Symbol (t+0)	Y
	Bits (7–0), Symbol (t+0)	Z
dout0	Source non-image packets from port	din0 and din1
	Halve control packet width	Off
	Color planes in parallel	2
	Color planes in sequence	2
	Bits (15–8), Symbol (t+0)	R
	Bits (15–8), Symbol (t+1)	X
	Bits (7–0), Symbol (t+0)	Y
	Bits (7–0), Symbol (t+1)	G
dout1	Port enabled	Off

The second example uses a Color Plane Sequencer to split an Avalon-ST video containing 4:2:2 subsampled data (Y'CbCr) into separate luminance (Y') and chrominance (Cb,Cr) streams (Figure 3–23 on page 3–28).

For this example, set the parameters listed in Table 3–21 in the **Parameter Settings** page.

Table 3–21. Parameters for the Color Plane Sequencer (Splitting Streams Example) (Part 1 of 2)

	Parameter	Value
	Bits per pixel per color plane	8
	Two pixels per port	On *(Note 1)*
din0	Color planes in parallel	1
	Color planes in sequence	4
	Bits (7–0), Symbol (t+0)	Cb
	Bits (7–0), Symbol (t+1)	Y
	Bits (7–0), Symbol (t+2)	Cr
	Bits (7–0), Symbol (t+3)	Y
din1	Port enabled	Off
dout0	Source non-image packets from port	din0
	Halve control packet width	On *(Note 2)*
	Color planes in parallel	1
	Color planes in sequence	2

Table 3–21. Parameters for the Color Plane Sequencer (Splitting Streams Example) (Part 2 of 2)

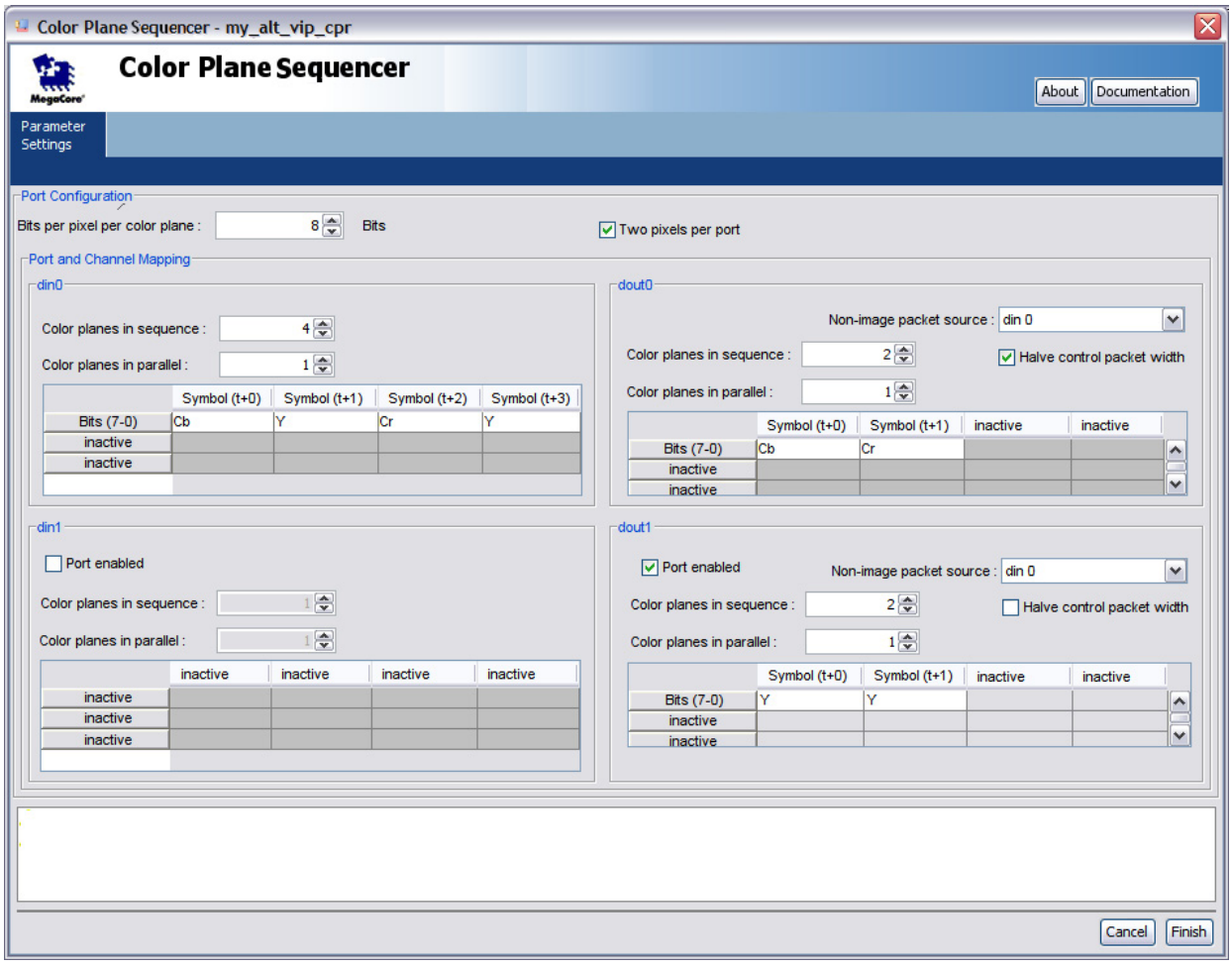
	Parameter	Value
	Bits (7–0), Symbol (t+0)	Cb
	Bits (7–0), Symbol (t+1)	Cr
dout1	Port enabled	On
	Source non-image packets from port	din0
	Halve control packet width	Off
	Color planes in parallel	1
	Color planes in sequence	2
	Bits (7–0), Symbol (t+0)	Y
	Bits (7–0), Symbol (t+1)	Y

Notes to Table 3–21:

(1) **Two pixels per port** needs to be On because two pixels worth of data is required to treat Cb and Cr separately. Alternatively, you can turn this parameter Off and use channel names C, Y instead of Cb, Y, Cr, Y.

(2) **Halve control packet width** needs to be On because this stream contains two subsampled channels. For other MegaCore functions to be able to treat these channels as two fully sampled channels in sequence, the control packet width must be halved.

Figure 3–23. Parameter Settings for the Color Plane Sequencer (Splitting Streams Example)



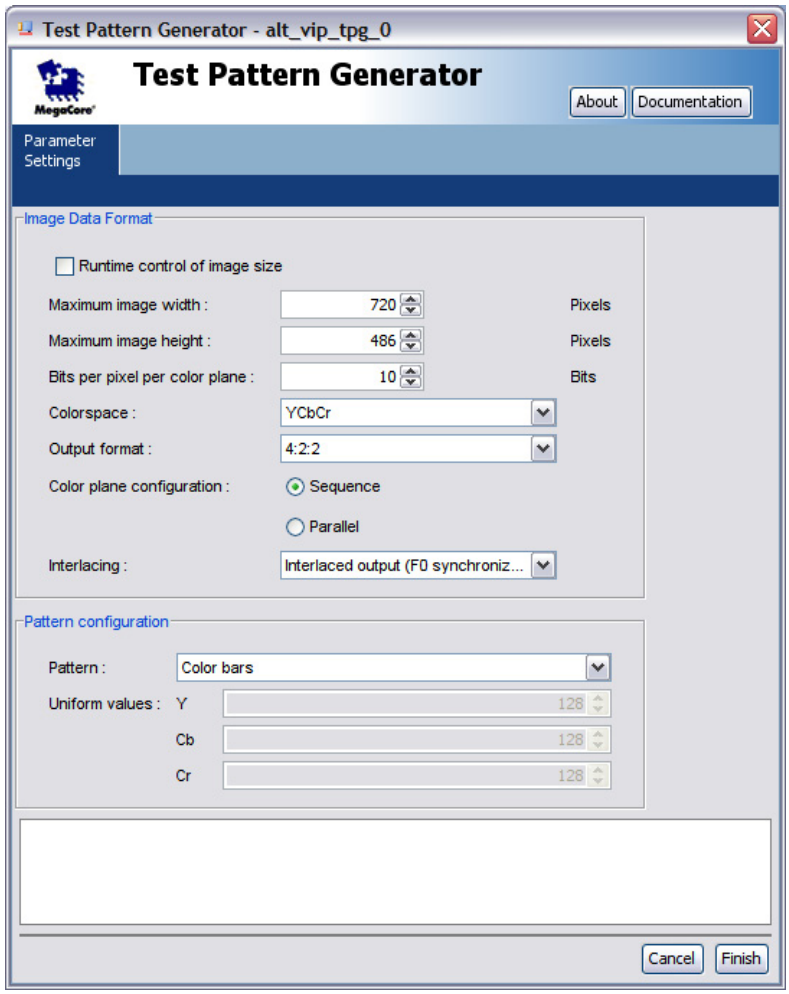
For more information about the options on the Color Plane Sequencer **Parameter Settings** page, refer to Table A–18 on page A–14.

Test Pattern Generator

The Test Pattern Generator MegaCore function generates a set of color bars for use as a test pattern.

Figure 3–11 shows the MegaWizard interface for the Test Pattern Generator.

Figure 3–24. Resolution Parameter Settings for the Test Pattern Generator



For example, to generate still video fields in a format emulating a NTSC video input, set the parameters listed in Table 3–22 in the **Parameter Settings** page.

Table 3–22. Parameters for the Test Pattern Generator

Parameter	Value
Run-time control of image size	Off
Maximum image width	720
Maximum image height	486

Table 3–22. Parameters for the Test Pattern Generator

Parameter	Value
Bits per pixel per color plane	10
Color space	YCbCr
Output format	4:2:2
Color plane configuration:	Sequence
Interlacing	Interlaced output (F0 synchronized) *(Note 1)*
Pattern	Color bars
Uniform values	Off *(Note 2)*

Note to Table 3–22:

(1) For this example, you can select either **F0 synchronized**, or **F1 synchronized** for the interlacing method.

(2) The options to specify uniform color values are only available when a **Uniform** pattern is selected.

For more information about the options on the **Resolution** tab of the Test Pattern Generator **Parameter Settings** page, refer to Table A–19 on page A–15.

Interface Types

The MegaCore® functions in the Video and Image Processing Suite use standard interfaces for data input and output, control input, and random access to external memory. These standard interfaces ensure that video systems can be quickly assembled by connecting MegaCores functions together, and facilitate the use of Altera system level design tools such as SOPC Builder.

The following types of interface are used:

- Avalon Streaming (Avalon-ST) interfaces using the Avalon Streaming Video protocol (Avalon-ST Video). The video protocol is used to transmit packets of video data and packets of control information in and out of the Video and Image Processing Suite MegaCore functions. This is the main method provided for connecting the MegaCore functions together to form image processing datapaths.
- Avalon Memory-Mapped (Avalon-MM) slave interfaces. These interfaces provide a means to change the image processing function being performed at run time.
- Avalon-MM master interfaces. These interfaces are used where the MegaCore functions require external memory.

Refer to the *Avalon Interface Specifications* for more information about these interface types.

These three interface types cover all of the data input and output requirements for eleven of the fourteen MegaCore functions in the Video and Image Processing Suite. The exceptions are the Line Buffer Compiler, Clocked Video Input, and Clocked Video Output MegaCore functions which use a lower level interface. For information about the interfaces these MegaCore functions use, refer to the functional description of the "Line Buffer Compiler" on page 5–28, "Clocked Video Input" on page 5–30, and "Clocked Video Output" on page 5–35.

Avalon-ST Video Protocol

The Avalon-ST Video protocol is a packet-oriented way to send video and control data over Avalon-ST connections. It defines two types of packets and allows for the definition of further packet types by users and by Altera.

The first type of packet contains a frame of video data that has been serialized into a stream of symbols. The second type of packet defines control information that applies to the subsequent video packet. An Avalon-ST connection carries a mix of packet types, and is required to match every video data packet with a preceding control packet.

When unrecognized packet types are sent, Avalon-ST Video requires that these packets are propagated unchanged and in the same order as they were received. This allows extended packet types to be freely mixed with the two basic packets types defined here.

The Avalon-ST Video protocol uses a subset of the signals defined by the *Avalon Interface Specifications*. Flow-controlled data transfers are made using the ready, valid, and data signal types and the boundaries between packets are marked using the startofpacket and endofpacket signal types.

The first value in each packet contains a 4-bit packet-type identifier. Table 4–1 lists the packet types. This value arrives in parallel with the startofpacket signal going high. Figure 4–2 on page 4–4 illustrates the transfer of a video data packet.

Table 4–1. Avalon-ST Video Packet Types

Type Identifier	Description
0	Video data
1–8	User packet types
9–14	Reserved for future Altera use
15	Control packet

Video Data Packets

There are many types of video data, differing in for example, resolution, interlacing, color spaces, and bit depths. There are also many ways to transmit the same video data, but there is no best way because different applications place differing priorities on factors such as cost, performance, and external memory bandwidth.

Avalon-ST Video data packets can be parameterized in a flexible way, allowing you to select the most appropriate format for a given application. The format of a video data packet is dictated by two sets of parameters. Each data packet contains either a field or a frame of video, depending on how these parameters are set.

The first set of parameters are provided when a video system is being constructed and cannot vary at run time. These static parameters define the shape of the Avalon-ST interfaces. Table 4–2 lists the static parameters and gives some examples of how they can be used.

Table 4–2. Examples of Static Avalon-ST Video Packet Parameters

Parameters		Description
Bits per Color Sample	Color Pattern	
8		Three color planes, R', G', and B' are transmitted in alternating sequence and each R', G', or B' sample is represented using 8 bits of data.
10		Three color planes are transmitted in parallel, leading to higher throughput than when transmitted in sequence, usually at higher cost. Each R', G', or B' sample is represented using 10 bits of data, so that, in total, 30 bits of data are transmitted in parallel.
10		4:2:2 video in the Y'CbCr color space, where there are twice as many Y' samples as Cb or Cr samples. One Y' sample and one of either a Cb or a Cr sample is transmitted in parallel. Each sample is represented using 10 bits of data. This is an example of PAL television transmitted according to the BT.656 standard.

Control packets provide the second set of parameters at run time (for MegaCore functions that support run-time reconfiguration) or when constructing the system. These dynamic parameters describe the format of the video frames traveling on the stream.

Table 4–3 lists the dynamic parameters and gives some examples of how they can be used. When these parameters are sent in a control packet, they apply to the next video data packet to arrive. When they are specified at compile-time, they apply to the first video data packet received after a reset. The table headings Frame Width and Frame Height are used for simplicity, but if the Interlacing flag is set such that the video is interlaced, these values actually define the field width and field height.

Table 4–3. Examples of Dynamic Avalon-ST Video Packet Parameters

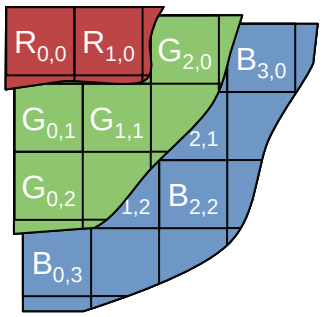
Parameters			Description
Frame Width	Frame Height	Interlacing	
640	480	Progressive	640×480 frames that are not interlaced.
1920	540	Field 0, Synch1	The following data is an interlaced field that is 1920 pixels wide and contains 540 lines, that is its frames are 1920×1080. The first field to arrive is *f0* (containing lines 0, 2, 4, ...). The field after that is *f1* (containing lines 1, 3, 5, ...). Subsequent fields alternate between *f0* and *f1*. The fields are synchronized on *f1*, that is a frame is formed by deinterlacing an *f0* field with the following *f1* field.
720	288	Field 1 Synch1	The following data is an interlaced field that is 720 pixels wide and 288 pixels high, that is its frames are 720×576. The first field to arrive is *f1*, and the deinterlacing is synchronized on *f1*. This means that a deinterlacer receiving this stream must already have buffered the previous *f0*, or drop this *f1* to be able to deinterlace correctly.

Examples

Consider a video sequence comprising progressive frames of 640×480 pixels in full R'G'B' color. Each frame contains 480 lines, each of which contains 640 pixels. Each pixel has three color values associated with it, red, green and blue. Let Rx,y Gx,y and Bx,y be the red, green and blue components of the pixel at coordinates (x,y) with the origin at the top left of the frame, $0 \leq x < 640$ and $0 \leq y < 480$.

Figure 4–1 shows part of the top left corner of a frame with color samples labelled in this way. The three different color values for each pixel are shown as three superimposed planes.

Figure 4–1. Part of a R'G'B' Frame



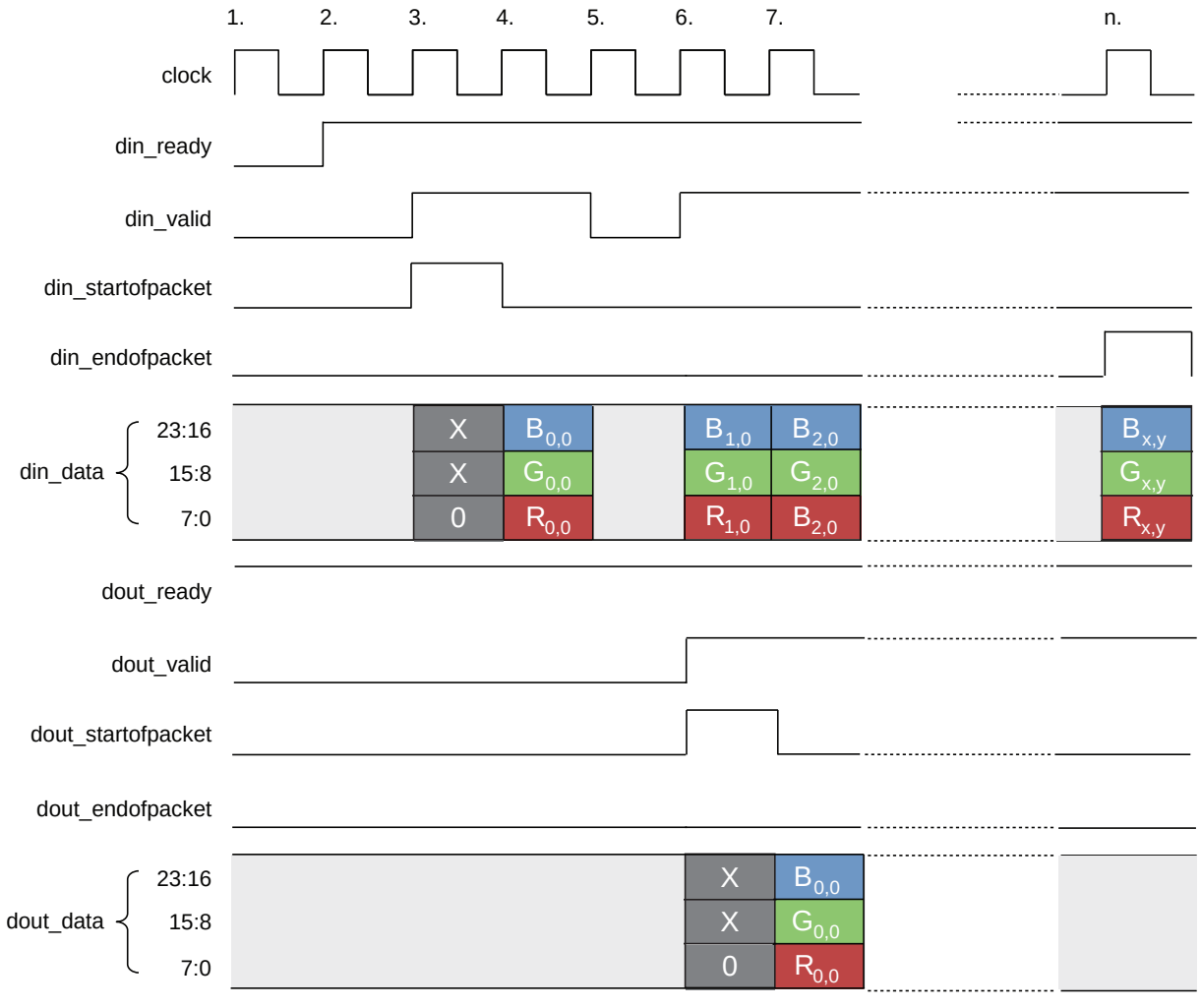
The number of binary bits that represents each color sample varies between different video systems. In the following examples, assume that the video data has eight bits per pixel per color plane. This means that all of the Rx,y Gx,y and Bx,y values for all x and y are eight bit values. The total number of bits used to represent each pixel is therefore 24.

Avalon-ST Video can describe different ways to transfer the same video data using different values of the color pattern parameter. For example, the R'G'B' video data described above could be transferred in parallel for maximum throughput or in an alternating sequence for reduced cost.

Data Transfer in Parallel

Figure 4–2 shows a timing diagram illustrating how the first few pixels of a frame in the video format from the preceding example might be processed by a MegaCore function which handles R'G'B' in parallel.

Figure 4–2. Timing Diagram Showing R'G'B' Transferred in Parallel



The example has one Avalon-ST port named `din` and one Avalon-ST port named `dout`. Data flows into the MegaCore function through `din`, is processed and flows out of the MegaCore function through `dout`.

There are five signals types (ready, valid, data, startofpacket, and endofpacket) associated with each port. The `din_ready` signal is an output from the MegaCore function and indicates when the input port is ready to receive data. The `din_valid` and `din_data` signals are both inputs. The source connected to the input port sets `din_valid` to logic '1' when `din_data` has useful information that should be sampled. `din_startofpacket` is an input signal that is raised to indicate the start of a packet, with `din_endofpacket` signaling the end of a packet.

The five output port signals have equivalent but opposite semantics.

The sequence of events shown in Figure 4–2 is:

1. Initially, `din_ready` is logic '0', indicating that the MegaCore function is not ready to receive data. Many of the Video and Image Processing Suite MegaCore functions are not ready for a few clock cycles in between rows of image data or in between video frames. For further details of each MegaCore function, refer to the "Functional Descriptions" on page 5–1.
2. The MegaCore function sets `din_ready` to logic '1', indicating that the input port is ready to receive data one clock cycle later. The number of clock cycles of delay which should be applied to a ready signal is referred to as ready latency in the *Avalon Interface Specifications*. All of the Avalon-ST interfaces used by the Video and Image Processing Suite have a ready latency of one clock cycle.
3. The source feeding the input port sets `din_valid` to logic '1' indicating that it is sending data on the data port and sets `din_startofpacket` to logic '1' indicating that the data is the first value of a new packet. The data itself is 0, indicating that the packet is video data.
4. The source feeding the input port holds `din_valid` at logic '1' and drops `din_startofpacket` indicating that it is now sending the body of the packet. It puts all three color values of the top left pixel of the frame on to `din_data`.
5. No data is transmitted for a cycle even though `din_ready` was logic '1' during the previous clock cycle and therefore the input port is still asserting that it is ready for data. This could be because the source has no data to transfer. For example, if the source is a FIFO, it could have become empty.
6. Data transmission resumes on the input port: `din_valid` transitions to logic '1' and the second pixel is transferred on `din_data`. Simultaneously, the MegaCore function begins transferring data on the output port. The example MegaCore function has an internal latency of three clock cycles so the first output is transferred three cycles after being received. This output is the type identifier for a video packet being passed along the datapath. For guidelines about the latencies of each Video and Image Processing MegaCore function, refer to the "Functional Descriptions" on page 5–1.
7. The third pixel is input and the first processed pixel is output.

n. For the final sample of a frame, the source sets `din_endofpacket` to logic '1', `din_valid` to '1', and puts the bottom-right pixel of the frame on to `din_data`.

In this example, both streams of pixel data have the Avalon-ST Video static parameters shown in Table 4–4.

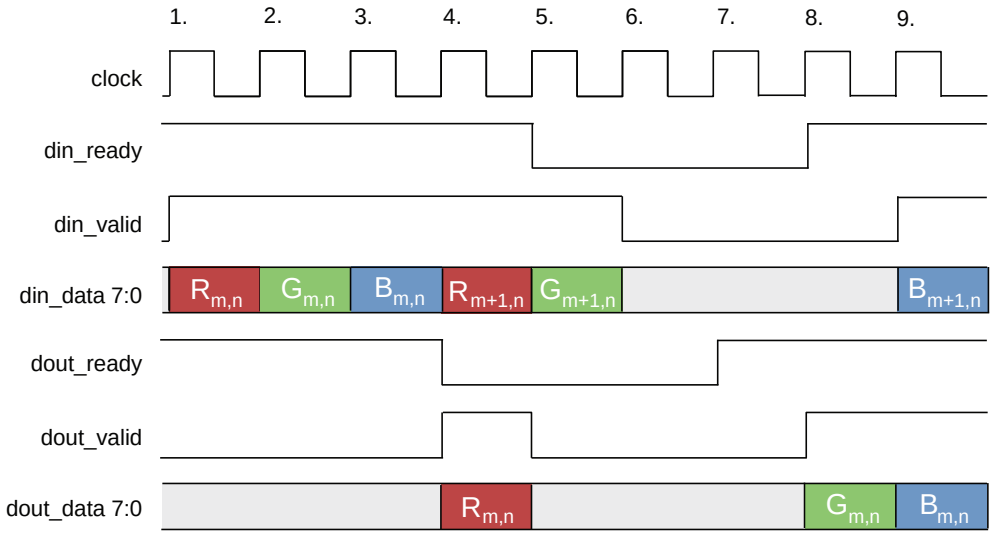
Table 4–4. Parameters for Example of Data Transferred in Parallel

Parameter	Value
Bits per Color Sample	8
Color Pattern	

Data Transfer in Sequence

Figure 4–3 shows a timing diagram illustrating how a number of pixels from the middle of a frame in the video format described on page 4–3 could be processed by another MegaCore function, this time handling R'G'B' in sequence.

Figure 4–3. Timing Diagram Showing R'G'B' Transferred in Sequence



Note to Figure 4–3:

(1) The `startofpacket` and `endofpacket` signals are not shown but are always low during the sequence shown in this figure.

This example is similar to Figure 4–2 on page 4–4 except that it is configured to accept R'G'B' data in sequence rather than parallel. The signals shown in the timing diagram are therefore the same but with the exception that the two data ports are only 8 bits wide.

The sequence of events shown in Figure 4–3 is:

1. Initially, `din_ready` is logic '1'. The source driving the input port sets `din_valid` to logic '1' and puts the red color value $R_{m,n}$ on the `din_data` port.
2. The source holds `din_valid` at logic '1' and the green color value $G_{m,n}$ is input.
3. The corresponding blue color value $B_{m,n}$ is input.

4. The MegaCore function sets `dout_valid` to logic '1' and outputs the red color value of the first processed color sample on the `dout_data` port. Simultaneously the sink connected to the output port sets `dout_ready` to logic '0'. The *Avalon Interface Specifications* state that sinks may set ready to logic '0' at any time, for example because the sink is a FIFO and it has become full.
5. The MegaCore function sets `dout_valid` to logic '0' and stops putting data on the `dout_data` port because the sink is not ready for data. The MegaCore function also sets `din_ready` to logic '0' because there is no way to output data and the MegaCore function must stop the source from sending more data before all internal buffer space is used up. The sink holds `din_valid` at logic '1' and transmits one more color sample $G_{m+1,n}$. This is legal because the ready latency of the interface means that the change in the MegaCore function's readiness does not take effect for one clock cycle.
6. Both the input and output interfaces transfer no data: the MegaCore function is stalled waiting for the sink.
7. The sink sets `dout_ready` to logic '1'. This could be because space has been cleared in a FIFO.
8. The MegaCore function sets `dout_valid` to logic '1' and resumes transmitting data. Now that the flow of data is again unimpeded, it sets `din_ready` to logic '1'.
9. The source responds to `din_ready` by setting `din_valid` to logic '1' and resuming data transfer.

In this example, both streams of pixel data have the Avalon-ST Video static parameters shown in Table 4–5.

Table 4–5. Parameters for Example of Data Transferred in Sequence

Parameter	Value
Bits per Color Sample	8
Color Pattern	R G B

Control Data Packets

Embedding control data packets in a stream of video data allows a video processing pipeline to reconfigure itself to new data as that data arrives. It simplifies or even eliminates the synchronization logic required to monitor the progress of changes through the datapath.

The basic Avalon-ST Video control packet contains a small but useful amount of information that applies to many application areas. The definitions of packet types listed in Table 4–1 on page 4–2 allow for other packet types to be created for targeting specific applications. It is required that every video data packet is preceded by a control data packet. To work with the smallest size of data port permitted by the Video and Image Processing Suite, these parameters are transmitted in 4-bit words. Four bits are used regardless of the size of video data being transmitted on a port so no conversion is necessary when the bit width changes along a processing pipeline.

User packets can use the entire bit width of the interface. However, care must be taken to handle truncation. For example, if a Color Space Converter changes from 10 bits to 8 bits, any 10-bit user packets are truncated.

Table 4–3 on page 4–3 lists examples of the parameters that are set by Avalon-ST Video control packets.

Each control packet contains the following 4-bit words in order:

```
type, width[15..12], width[11..8], width[7..4], width[3..0],
height[15, 12], height[11..8], height[7..4], height[3..0],
interlacing.
```

The type for a control packet is always 15. The width and height parameters are split across four words, and transmitted as unsigned integers with the most-significant bits first. The interlacing field is coded to indicate progressive data or which field is being sent and how fields should be used to reconstruct frames.

The most significant two bits of the interlacing word describe whether the data is either progressive, interlaced *f0* (contains lines 0, 2, 4, ...), or interlaced *f1* (contains lines 1, 3, 5, ...). 00 means progressive, 10 means interlaced *f0*, and 11 means interlaced f1.

The second two bits describe the synchronization of interlaced data. Synchronizing on f0 means that a frame should be constructed by deinterlacing an *f1* followed by an f0. Similarly, synchronizing on *f1* means that deinterlacing uses *f0* then *f1*. The encoding for this is `00` for synchronize on *f0*, `01` for synchronize on *f1*, and `11` for "don't care".

The Video and Image Processing Suite MegaCore functions are designed to propagate any unexpected data in the control packets, so that any future extensions are passed on without modification.

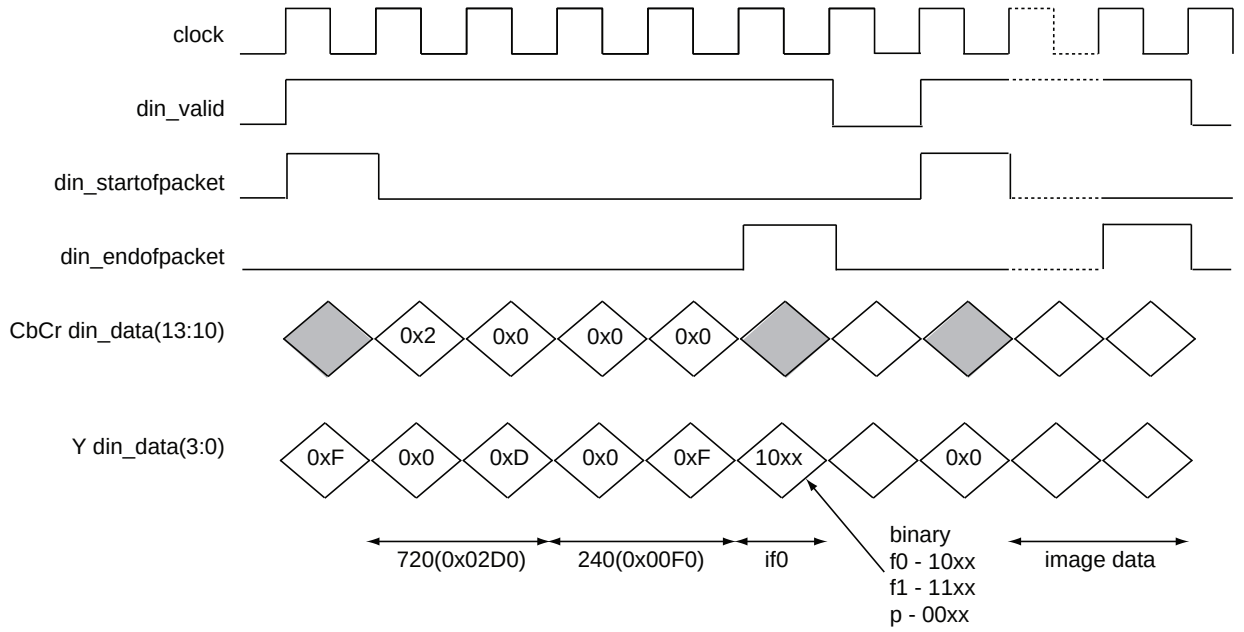
Table 4–6 lists some example control packets.

Table 4–6. Examples of Avalon-ST Video Control Packets

Type	Width	Height	Interlacing	Description
15	0000, 0010, 1000, 0000	0000, 0001, 1110, 0000	0000	The following frames are progressive with resolution of 640×480 pixels.
15	0000, 0010, 1000, 0000	0000, 0000, 1111, 0000	1000	The following fields are 640 pixels wide and 240 pixels high. The next field is *f0*, and frames of size 640×480 are constructed by deinterlacing *f1* with the following *f0*.
15	0000, 0111, 1000, 0000	0000, 0010, 0001, 1100	1100	The following fields are 1920 pixels wide and 540 pixels high. The next field is *f1*, and deinterlacing *f0* followed by *f1* will produce frames of size 1920×1080.

Figure 4–4 shows the transfer of a control packet for a field of 720×480i video (with field height 240). It is transferred over an interface configured for 10-bit data with 2 color planes in parallel. Each word of the control packet is transferred in the lowest four bits of a color plane, starting with bits 3:0, then 13:10.

Figure 4–4. ***Example of Control Packet Transfer***



Packet Propagation

The Avalon-ST Video protocol is designed to be optimized for the transfer of video data while still providing a flexible way to transfer control data and other information. To make the protocol flexible and extensible, the Video and Image Processing MegaCore functions obey the following rules about propagating non-video packets:

- Packets are propagated until the endofpacket signal is received, regardless of their expected length.
- When the bits per color sample change from the input to the output side of a block, the non-video packets are truncated or padded. Otherwise, the full bit width is transferred.
- When the color pattern changes from the input to the output side of a block, the symbol order is preserved and redundant data might need to be inserted. For example:

```
2
1 -> 012
0
```

Avalon-ST Video Specification

The Avalon-ST Video Specification comprises the following parts:

- "Avalon-ST Video Parameters" on page 4–10 describes the parameters Avalon-ST Video uses to describe a serialized stream of pixel data.
- "Type of Avalon-ST Interfaces Used" on page 4–11 details the particular type of Avalon-ST interface the protocol uses and shows how that interface type depends upon the parameters of the data transferred.
- "Avalon-ST Video Rules" on page 4–11 formally defines the rules governing how sequences of video images are serialized for transmission.

Avalon-ST Video Parameters

Table 4–7 lists the parameters used by Avalon-ST Video.

Table 4–7. Avalon-ST Video Parameters

Parameter Name	Description	Example
Static Parameters:		
Bits per Color Sample	Maximum number of binary bits used to represent each color sample.	8
Color Pattern	A matrix defining a repeating pattern of color samples to be transmitted. The height of the matrix indicates the number of samples transmitted in parallel, the width determines how many cycles of data are transmitted before the pattern repeats. In the common case, each element of the matrix contains the name of a color plane from which a sample should be taken. The exception is for vertically subsampled color planes. These are indicated by writing the names of two color planes in a single element, one above the other. Samples from the upper color plane are transmitted on even rows and samples from the lower plane are transmitted on odd rows.	
Dynamic Parameters:		
Frame Width	Width in pixels for the frames of the video stream. Must be a positive integer.	320
Frame Height	Height in pixels for the frames/fields in the video stream. Note that in the case of interlaced video streams, the height of frames after deinterlacing is double the value given here. Must be a positive integer.	240
Interlacing	A value of "Progressive" specifies that it is a progressive video stream. Interlaced streams are specified in two parts: ■ The field number (Field 0 or Field 1) indicates whether the next field is f0 or f1. ■ The field synchronization (Synch 0, Synch 1, or Don't care) indicates how to perform deinterlacing. Synch 0 means that deinterlacing is performed using f1 and it's following f0 to construct a frame. Likewise, Synch 1 uses f0 and its following f1. Don't care assumes that fields arrive in the correct order to be deinterlaced without dropping any fields.	Progressive

A set of values for these parameters can be used to describe any type of video data stream that can be transmitted according to the protocol. The example parameter values given in Table 4–7 describe a stream of progressive Y'CbCr 4:2:0 (horizontally and vertically subsampled) video in 320×240 resolution with 8 bits per color plane transmitted in sequence.

Type of Avalon-ST Interfaces Used

The *Avalon Interface Specifications* define parameters which can be used to specify any type of Avalon-ST interface. Table 4–8 lists the values of these parameters that are defined for the Avalon-ST interfaces used by the Video and Image Processing Suite MegaCore functions. All parameters not explicitly listed in the table have undefined values.

Table 4–8. Avalon-ST Interface Parameters

Parameter Name	Value
BITS_PER_SYMBOL	Variable. Always equal to the Bits per Color Sample parameter value of the stream of pixel data being transferred.
SYMBOLS_PER_BEAT	Variable. Always equal to the number of color samples being transferred in parallel. This is equivalent to the number of rows in the color pattern parameter value of the stream of pixel data being transferred.
READY_LATENCY	1

The *Avalon Interface Specifications* define many signal types many of which are optional. Table 4–9 lists the signals used by the Avalon-ST interfaces for the Video and Image Processing Suite MegaCore functions. Any signal type not explicitly listed in the table is not included.

Table 4–9. Avalon-ST Interface Signal Types

Signal	Width	Direction
`ready`	1	Sink to Source
`valid`	1	Source to Sink
`data`	`bits_per_symbol` × `symbols_per_beat`	Source to Sink
`startofpacket`	1	Source to Sink
`endofpacket`	1	Source to Sink

Avalon-ST Video Rules

This section specifies the Avalon-ST Video rules. These rules, combined with a set of video protocol parameters, define how MegaCore functions that are compatible with the protocol send and receive video data. A working definition of a video clip is given and the relationship between the Avalon-ST Video parameters and the type of video clip transmitted is described. A procedure is then shown for reconstructing a video clip from any Avalon-ST Video stream, given a set of parameter values. This procedure is the specification of the protocol rules for Avalon-ST Video.

A flexible definition of a video clip is required because there are many different types of video that can be transferred using Avalon-ST Video. A video clip is defined as an ordered sequence of frames. Each frame consists of either one field (progressive video) or two fields numbered 0 and 1 (interlaced video). Field 0 is the field that includes the top line of the frame. Each field contains a set of color planes. For example, R, G, and B color planes in full color R'G'B' video.

Because Avalon-ST Video supports various kinds of subsampled video data, the color planes of a field can be of differing sizes. For example, a 720×576 pixel frame of Y'CbCr 4:2:2 progressive video contains one field having three color planes, Y, Cb, and Cr. Because the video is 4:2:2 subsampled, only the Y plane is 720×576 samples in size and each of the Cb and Cr planes have 360×576 samples.

The Avalon-ST Video parameters define the type of video clip represented by a stream. The Interlaced/Progressive parameter defines whether the video clip is interlaced or progressive. The number of bits used for each sample is defined by the Bits per Color Sample parameter. The names, widths, and heights of each color plane of each field are derived from the Frame Width, Frame Height, and Color Pattern parameters as follows. For each color *C*, represented in the color pattern matrix:

$$Width_C = I_C \times R$$

where I_C is the number of times *C* appears in the color pattern, and *R* is an integer that is equal to the frame width divided by the maximum value of I_C for all color planes.

$Height_C = Frame\ \ Height$ if *C* is not vertically subsampled

$Height_C = \frac{Frame\ \ Height}{2}$ otherwise

For example, consider a stream of pixel data with the parameters shown in Table 4–10.

Table 4–10. Parameters for Video Clip Calculation Example

Parameter	Value
Frame Width	320
Frame Height	240
Interlaced/Progressive	Progressive
Bits per Color Sample	8
Color Pattern	

Three color planes are represented in the color pattern, Y, Cb, and Cr. Consequently, the video clip transmitted has the same three color planes. Y occurs twice in the pattern, Cb and Cr occur once each:

$I_Y = 2;$

$I_{Cb} = I_{Cr} = 1$

The value of *R* can be calculated using the occurrence of each plane and the Frame Width parameter:

$$R = \frac{Frame\ \ Width}{\max(I_Y, I_{Cb}, I_{Cr})} = \frac{320}{2} = 160$$

Using I_Y, I_{Cb}, I_{Cr}, and *R*, you can calculate the width of each color plane:

$Width_Y = I_Y \times R = 2 \times 160 = 320$

$Width_{Cb} = I_{Cb} \times R = 1 \times 160 = 160$

$Width_{Cr} = 1_{Cr} \times R = 1 \times 160 = 160$

You can also calculate the height of each of the three color planes. The stream is progressive, so the number of fields per frame is 1.

Plane Y is shown as not being vertically subsampled in the color pattern:

$$Height_Y = 240$$

The two color difference planes, Cb and Cr, appear as a vertically subsampled pair in the color pattern:

$$Height_{Cb} = \frac{Frame\ Height}{2} = \frac{240}{2} = 120$$

The stream described by the parameter values in this example therefore describes a video clip containing three color planes named Y, Cb, and Cr, where the Y plane is 320×240 pixels, and the Cb and Cr planes are both 160×120 pixels, a quarter the size of the Y plane. This is consistent with 4:2:0 format video.

Video clips can be reconstructed by repeatedly applying the same process to construct one frame at a time. The process can be considered in two stages:

1. Input data is read and split it into a set of ordered sequences C_S, of samples for each color, C using the information held in the Color Pattern:

```
for (0 ≤ y < Frame Height)
   for (0 ≤ x < R)
      for (0 ≤ i < width of Color Pattern)
      D ← next word of data
         for (0 ≤ j < height of Color Pattern)
            E ← Color Pattern(i, j)
            if E is vertically subsampled then
               C ← top(E) if y is even, bottom(E)
            else
               C ← plane(E)
            end if
            append D(j) to C_S
         end for
      end for
   end for
end for
```

2. A video frame is then constructed using the sequences of color samples C_S:

```
Frame ← create a new empty frame
   for (0 ≤ f < Number of Fields)
   Field ← create a new empty field in Frame
   for each color C, represented in Color Pattern
      Color Plane ← new plane of size Width_C × Height_C
      for j in 0 to Height_C
         for i in 0 to Width_C
            Color Plane (i, j) ← next element of C_S
         end for
      end for
   end for
end for
```

In this code, *Color Pattern [i, j]* refers to the element of the color pattern matrix found at the intersection of column *i* and row *j*, where column zero is the far left column and row zero is the top row. This element *E* can be vertically subsampled, in which case top(*E*) selects the color plane written in the top half of the element and bottom(*E*) selects the color plane written below it. If the element is not vertically subsampled, then plane(*E*) refers to the color plane in the element.

D(j) refers to symbol j in data word *D* read from an Avalon-ST interface. According to the *Avalon Interface Specifications*, symbol 0 occupies the most significant bits in the data word, symbol 1 is next and so on.

Avalon-MM Slave Interfaces

The Video and Image Processing Suite MegaCore functions that permit run-time control of some aspects of their behavior, use a common type of Avalon-MM slave interface for this purpose.

Each slave interface provides access to a set of control registers which must be set by external hardware. You should 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.

For a full description of the control registers, refer to "Run-Time Control Register Maps" on page A–16.

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, all other bits are unused. A few cycles after the MegaCore 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).

  The MegaCore function does not process any data until the `Go` bit is set by external logic connected to the control port. This allows run-time control data to be programmed before the processing begins. A few cycles after `Go` is set, the MegaCore function begins processing data. If the `Go` bit is unset while the data is being processed, then it stops processing data again at the end of the current video frame, and waits until the `Go` bit is set again by external logic.

- Register 1 is the `Status` register. Bit zero of this register is the `Status` bit, all other bits are unused. The MegaCore function sets the `Status` bit to 1 when it is running, and zero otherwise. External logic attached to the control port should not attempt to write to the `Status` register.

The following pseudo-code illustrates the design of MegaCore functions that double-buffer their control (that is, all MegaCore functions except the Gamma Corrector and some Scaler 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();
}
```

The MegaCore function reads packets from the input stream and processes them until the image data header (0) of an image data packet has been received. There is a small amount of buffering at the input of each Video and Image Processing Suite MegaCore function and you should expect that most functions read a few samples past the image data header if data is available. These samples are stored and are processed as usual when the status bit is set back to 1.

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 MegaCore function 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 MegaCore function to stop processing at the end of the current frame.
2. Poll the `Status` bit until the MegaCore function sets it to zero. This occurs at the end of the current frame, after the MegaCore function has stopped processing data.
3. Update the gamma look-up table.
4. Set the `Go` bit to one. This causes the MegaCore function to start processing the next frame.
5. Poll the `Status` bit until the MegaCore function sets it to one. This occurs when the MegaCore function 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 MegaCore function is not processing data while the update is performed. When using MegaCore functions which double-buffer control data, such as the Alpha Blending Mixer and Scaler, a more simple process may be sufficient:

1. Set the `Go` bit to zero. This causes the MegaCore function 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 MegaCore function.

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.

Specification of the Type of Avalon-MM Slave Interfaces Used

The *Avalon Interface Specifications* define many signal types, many of which are optional.

Table 4–11 lists the signals used by the Avalon-MM slave interfaces in the Video and Image Processing Suite. Any signal type that is not explicitly listed in the table is not used.

Table 4–11. Avalon-MM Slave Interface Signal Types

Signal	Width	Direction
`chipselect`	1	Input
`address`	Variable	Input
`readdata`	Variable	Output
`write`	1	Input
`writedata`	Variable	Input

Clock and reset signal types are not included. The Video and Image Processing Suite does not support Avalon-MM interfaces in multiple clock domains. Instead, all of the Video and Image Processing Suite MegaCore functions have one clock input and one reset input. The Avalon-MM slave interfaces must operate synchronously to this clock.

The *Avalon Interface Specifications* define a set of transfer properties which may or may not be exhibited by any Avalon-MM interface. Together with the list of supported signals, these properties fully define an interface type.

The control interfaces of the Video and Image Processing Suite MegaCore functions exhibit the following transfer properties:

- Zero wait states on write operations
- Two wait states on read operations"Run-Time Control Register Maps" on page A–16"Run-Time Control Register Maps" on page A–16Table 4–11

Avalon-MM Master Interfaces

The Video and Image Processing Suite MegaCore functions use a common type of Avalon-MM master interface for access to external memory. These master interfaces should be connected to external memory resources via arbitration logic such as that provided by the system interconnect fabric.

Specification of the Type of Avalon-MM Master Interfaces Used

The *Avalon Interface Specifications* define many signal types, many of which are optional.

Table 4–12 on page 4–17 lists the signals used by the Avalon-MM master interfaces in the Video and Image Processing Suite. Any signal type not explicitly listed in the table is not used.

Table 4–12. Avalon-MM Master Interface Signal Types

Signal	Width	Direction	Usage
`clock`	1	Input	Read-Write (optional)
`readdata`	variable	Input	Read-only
`readdatavalid`	1	Input	Read-only
`reset`	1	Input	Read-Write (optional)
`waitrequest`	1	Input	Read-write
`address`	32	Output	Read-write
`burstcount`	variable	Output	Read-write
`read`	1	Output	Read-only
`write`	1	Output	Write-only
`writedata`	variable	Output	Write-only

The clock and reset signal types are optional. The Avalon-MM master interfaces can operate on a different clock from the MegaCore function and its other interfaces by selecting the relevant option in the MegaWizard interface when and if it is available.

Some of the signals in Table 4–12 are read-only and not required by a master interface which only performs write transactions.

Some other signals are write-only and not required by a master interface which only performs read transactions. To simplify the Avalon-MM master interfaces and improve efficiency, read-only ports are not present in write-only masters, and write-only ports are not present in read-only masters.

Read-write ports are present in all Avalon-MM master interfaces. Refer to the description of each MegaCore function for information about whether the master interface is read-only, write-only or read-write.

The *Avalon Interface Specifications* define a set of transfer properties which may or may not be exhibited by any Avalon-MM interface. Together with the list of supported signals, these properties fully define an interface type.

The external memory access interfaces of the Video and Image Processing Suite MegaCore functions exhibit the following transfer property:

- Pipeline with variable latency

Buffering of Non-Image Data Packets in Memory

The Frame Buffer and the Deinterlacer (when buffering is enabled) route the video stream through an external memory. Avalon-ST Video control packets and user packets must be buffered and delayed along with the frame or field they relate to and extra memory space has to be allocated. You must specify the maximum number of packets per field and the maximum size of each packet to cover this requirement.

The maximum size of a packet is given as a number of symbols, header included. For instance, the size of an Avalon-ST Video control packet is 10. This size does not depend on the number of channels transmitted in parallel. Packets larger than this maximum limit may be truncated as extra data is discarded.

The maximum number of packets is the number of packets that can be stored with each field or frame. Older packets are discarded first in case of overflow. When frame dropping is enabled, the packets associated with a field that has been dropped are automatically transferred to the next field and count towards this limit.

Altera recommends that you keep the default values for **Number of packets buffered per frame** = 1 and **Maximum packet length** = 10, unless you intend to extend the Avalon-ST Video protocol with custom packets. Using the default parameterization, the relevant MegaCore functions may overwrite old control packets with newer ones. This does not matter because the last control packet always takes precedence.

Each Video and Image Processing MegaCore function is implemented to generate hardware that performs its operations on multiple color planes (typically three).

Color Space Converter

The Color Space Converter MegaCore function provides a flexible and efficient means to convert image data from one color space to another.

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 may need 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.

Input and Output Data Types

The Color Space Converter MegaCore function inputs and outputs support signed or unsigned data and 4 to 20 bits per pixel per color plane. Minimum and maximum guard bands are also supported. The guard bands specify ranges of values that should never be received by, or transmitted from the MegaCore function. Using output guard bands allows the output to be constrained, such that it does not enter the guard bands.

Color Space Conversion

Conversions between color spaces are achieved by providing an array of nine coefficients and three summands that relate the color spaces. These can be set at compile time, or at run time using the Avalon-MM slave interface.

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 output values on channels 0, 1, and 2 (denoted dout_0, dout_1, and dout_2) are calculated as follows:

$$dout\_0 = (A0 \times din\_0) + (B0 \times din\_1) + (C0 \times din\_2) + S0$$
$$dout\_1 = (A1 \times din\_0) + (B1 \times din\_1) + (C1 \times din\_2) + S1$$
$$dout\_2 = (A2 \times din\_0) + (B2 \times din\_1) + (C2 \times din\_2) + S2$$

where din_0, din_1, and din_2 are inputs read from channels 0, 1, and 2 respectively.

User-specified custom constants and the following predefined conversions are supported:

- 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 is being used to input 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.

Predefined conversions only support unsigned input and output data. If signed input or output data is selected, the predefined conversion produces incorrect results. When using a predefined conversion, the precision of the constants must still be defined. Predefined conversions are based on the input bits per pixel per color plane. If using different input and output bits per pixel per color plane, the results should be scaled by the correct number of binary places to compensate.

Constant Precision

The Color Space Converter MegaCore function requires fixed point types to be defined for the constant coefficients and constant summands. The user entered constants (in the white cells of the matrix in the MegaWizard interface) are rounded to fit in the chosen fixed point type (these are shown in the purple cells of the matrix).

Calculation Precision

The Color Space Converter MegaCore function does not lose calculation precision during the conversion. The calculation and result data types are derived from the range of the input data type, the fixed point types of the constants, and the values of the constants. If scaling is selected, the result data type is scaled up appropriately such that precision is not lost.

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 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. There are three methods:
   - 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. There are two 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 the range specified by the output data type (output guard bands, or if unspecified the minimum and maximum values allowed by the output bits per pixel), logic that saturates the results to the minimum and maximum output values is automatically added.

The Color Space Converter MegaCore function can process streams of pixel data of the types shown in Table 5–1.

Table 5–1. Color Space Converter Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	Read from control packets at run time.
Frame Height	Read from control packets at run time.
Interlaced/Progressive	Either.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color Pattern	For color planes in sequence:  For color planes in parallel: 

Notes to Table 5–1:

(1) For channels in parallel, the top of the color pattern matrix represents the MSB of data and the bottom represents the LSB. For details, refer to "Avalon-ST Video Specification" on page 4–10.

Chroma Resampler

The Chroma Resampler MegaCore function allows you to change between 4:4:4, 4:2:2 and 4:2:0 sampling rates where:

- 4:4:4 specifies full resolution in planes 1, 2, and 3
- 4:2:2 specifies full resolution in plane 1; half width resolution in planes 2 and 3
- 4:2:0 specifies full resolution in plane 1; half width and height resolution in planes 2 and 3

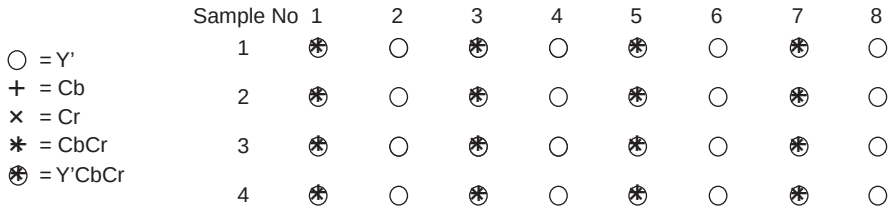
All modes of the Chroma Resampler assume the chrominance (chroma) and luminance (luma) samples are co-sited (that is, their values are sampled at the same time). The horizontal resampling process supports nearest-neighbor and filtered algorithms. The vertical resampling process only supports the nearest-neighbor algorithm.

The Chroma Resampler MegaCore function can be configured to change image size at run time using control packets.

Horizontal Resampling (4:2:2)

Figure 5–1 shows the location of samples in a co-sited 4:2:2 image.

Figure 5–1. Resampling 4.4.4 to a 4.2.2 Image



Conversion from sampling rate 4:4:4 to 4:2:2 and back are scaling operations on the chroma channels. This means that these operations are affected by some of the same issues as the Scaler MegaCore function. However, because the scaling ratio is fixed as 2× up or 2× down, the Chroma Resampler MegaCore function is highly optimized for these cases.

The Chroma Resampler MegaCore Function only supports the co-sited form of horizontal resampling. This is the form used for 4:2:2 data in *ITU Recommendation BT.601*, MPEG-2, and other standards.

For more information about the ITU standard, refer to *Recommendation ITU-R BT.601, Encoding Parameters of Digital Television for Studios, 1992, International Telecommunications Union, Geneva*.

4:4:4 to 4:2:2

The nearest-neighbor algorithm is the simplest way to down-scale the chroma channels. It works by simply discarding the Cb and Cr samples that occur on even columns (assuming the first column is numbered 1). This algorithm is very fast and cheap but, due to aliasing effects, it does not produce the best image quality.

To get the best results when down-scaling, you can apply a filter to remove high-frequency data and thus avoid possible aliasing. The filtered algorithm for horizontal subsampling uses a 9-tap filter with a fixed set of coefficients.

The coefficients are based on a Lanczos-2 function (Refer to "Choosing and Loading Coefficients" on page 5–17) as used by the Scaler MegaCore function, and their quantized form is known as the Turkowski Decimator.

For more information about the Turkowski Decimator, refer to *Ken Turkowski. Graphics Gems, chapter Filters for common resampling tasks, pages 147–165. Academic Press Professional, Inc., San Diego, CA, USA, 1990.*

Because the coefficients are fixed and approximate to powers of two, they can be implemented by bit-shifts and additions. This algorithm efficiently eliminates aliasing in the chroma channels, and uses no memory or multipliers. However, it does use more logic area than the nearest-neighbor algorithm.

4:2:2 to 4:4:4

The nearest-neighbor algorithm is the simplest way to up-scale the chroma channels. It works by simply duplicating each incoming Cb and Cr sample to fill in the missing data. This algorithm is very fast and cheap but it tends to produce sharp jagged edges in the chroma channels.

The filtered algorithm uses the same method as the Scaler MegaCore function would use for upscaling, that is a four-tap filter with Lanczos-2 coefficients. This filter is used with a phase offset of 0 for the odd output columns (those with existing data) and an offset of one-half for the even columns (those without direct input data). A filter with phase offset 0 has no effect, so it is implemented as a pass-through. A filter with phase offset of one-half interpolates the missing values and has fixed coefficients that are implemented by bit-shifts and additions.

This algorithm does a reasonable job at upsampling and uses no memory or multipliers. It uses more logic elements than the nearest-neighbor algorithm and is not the highest quality available.

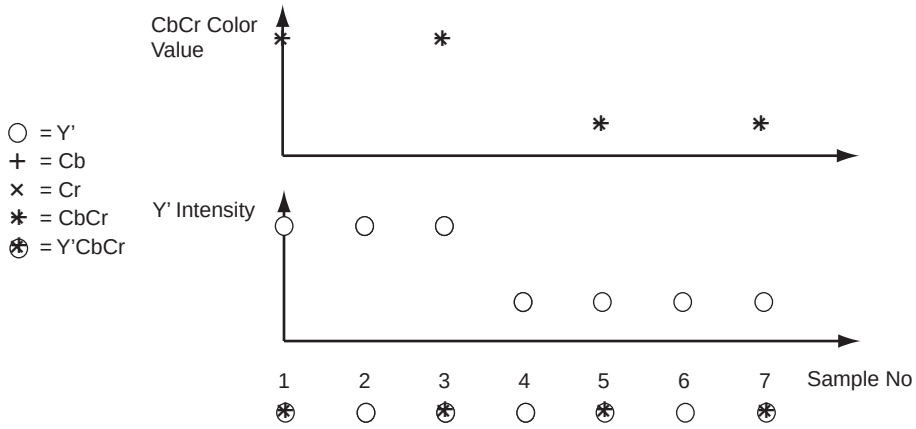
The best image quality for upsampling is obtained by using the filtered algorithm with luma-adaptive mode enabled. This mode looks at the luma channel during interpolation and uses this to detect edges. Edges in the luma channel are used to make appropriate phase-shifts in the interpolation coefficients for the chroma channels.

Figure 5–2 on page 5–6 shows 4:2:2 data at an edge transition. Without taking any account of the luma, the interpolation to produce chroma values for sample 4 would weight samples 3 and 5 equally. From the luma, you can see that sample 4 falls on an the low side of an edge, so sample 5 is more significant than sample 3.

The luma-adaptive mode looks for such situations and chooses how to adjust the interpolation filter. From phase 0, it can shift to -1/4, 0, or 1/4; from phase 1/2, it can shift to 1/4, 1/2, or 3/4. This makes the interpolated chroma samples line up better with edges in the luma channel and is particularly noticeable for bold synthetic edges such as text.

The luma-adaptive mode uses no memory or multipliers, but requires more logic elements than the straightforward filtered algorithm.

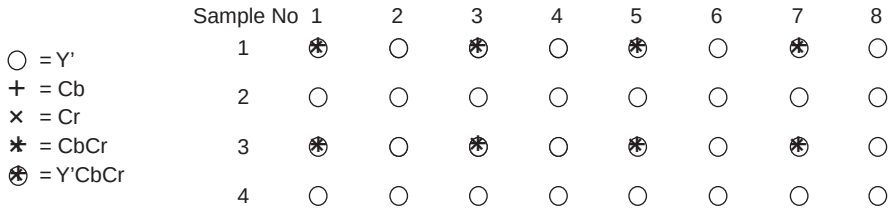
Figure 5–2. 4:2:2 Data at an Edge Transition



Vertical Resampling (4:2:0)

The Chroma Resampler MegaCore function does not distinguish interlaced data with its vertical resampling mode. It only supports the co-sited form of vertical resampling shown in Figure 5–3.

Figure 5–3. Resampling 4.4.4 to a 4.2.0 Image



For both upsampling and downsampling, the vertical resampling algorithm is fixed at nearest-neighbor.

Vertical resampling does not use any multipliers. For upsampling, it uses four line buffers, each buffer being half the width of the image. For downsampling it uses one line buffer which is half the width of the image.

All input data samples must be in unsigned format. If the number of bits per pixel per color plane is *N*, this means that each sample consists of *N* bits of data which are interpreted as an unsigned binary number in the range [0, 2^N – 1]. All output data samples are also in the same unsigned format.

For more information about how non-video packets are transferred, refer to "Packet Propagation" on page 4–9.

The Chroma Resampler MegaCore function can process streams of pixel data of the types shown in Table 5–2.

Table 5–2. Chroma Resampler Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	Maximum frame width is specified in the MegaWizard interface, the actual value is read from control packets.
Frame Height	Maximum frame height is specified in the MegaWizard interface, the actual value is read from control packets.
Interlaced/Progressive	Progressive.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color Pattern	For 4:4:4 sequential data:  For 4:2:2 sequential data:  For 4:2:0 sequential data:  For 4:2:2 parallel data:  For 4:4:4 parallel data:  For 4:2:0 parallel data: 

Gamma Corrector

The Gamma Corrector MegaCore function provides a look-up table (LUT) accessed through an Avalon-MM slave port. The gamma values can be entered in the LUT by external hardware using this interface.

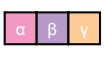
For information about using Avalon-MM slave interfaces for run-time control in the Video and Image Processing Suite, refer to "Avalon-MM Slave Interfaces" on page 4–14. For details of the control register maps, refer to Table A–21 on page A–17, Table A–22 on page A–17, and Table A–23 on page A–17. For information about the Avalon-MM interface signals, refer to Table A–38 on page A–28.

When dealing with image data with N bits per pixel per color plane, the address space of the Avalon-MM slave port spans $2^N + 2$ registers where each register is N bits wide.

Registers 2 to $2^N + 1$ are the look-up values for the gamma correction function. Image data with a value x will be mapped to whatever value is in the LUT at address $x + 2$.

The Gamma Corrector MegaCore function can process streams of pixel data of the types shown in Table 5–3.

Table 5–3. Gamma Corrector Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	Read from control packets at run time.
Frame Height	Read from control packets at run time.
Interlaced/Progressive	Either.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color Pattern	One, two or three channels in sequence or parallel. For example, if three channels in sequence is selected where a, b and g can be any color plane: 

2D FIR Filter

The 2D FIR Filter performs 2D convolution, using matrices of 3×3, 5×5, and 7×7 coefficients.

The MegaCore function retains full precision throughout the calculation, while making efficient use of FPGA resources. With suitable coefficients, the MegaCore function can perform several operations including, but not limited to sharpening, smoothing and edge detection.

An output pixel is calculated from the multiplication of input pixels in a filter size grid (kernel) by their corresponding coefficient in the filter.

These values are summed together. Prior to output, this result is scaled, has its fractional bits removed, is converted to the desired output data type, and is constrained to a specified range. The position of the output pixel corresponds to the mid-point of the kernel. If the kernel runs over the edge of an image, then zeros are used for the out of range pixels.

The 2D FIR Filter allows its input, output and coefficient data types to be fully defined. Constraints are 4 to 20 bits per pixel per color plane for input and output, and up to 35 bits for coefficients.

The 2D FIR Filter supports symmetric coefficients. This reduces the number of multipliers, resulting in smaller hardware. Coefficients can be set at compile time, or changed at run time using an Avalon-MM slave interface.

Calculation Precision

The 2D FIR Filter does not lose calculation precision during the FIR calculation. The calculation and result data types are derived from the range of input values (as specified by the input data type, or input guard bands if provided), the coefficient fixed point type and the coefficient values. If scaling is selected, then the result data type is scaled up appropriately such that precision is not lost.

Coefficient Precision

The 2D FIR Filter requires a fixed point type to be defined for the coefficients. The user-entered coefficients (shown as white boxes in the MegaWizard interface) are rounded to fit in the chosen coefficient fixed point type (shown as purple boxes in the MegaWizard interface).

Result to 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 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.

There are three methods:

- 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. There are two 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 the range specified by the output data type (output guard bands, or if unspecified the minimum and maximum values allowed by the output bits per pixel), logic to saturate the results to the minimum and maximum output values is automatically added.

The 2D FIR Filter MegaCore function can process streams of pixel data of the types shown in Table 5–4.

Table 5–4. 2D FIR Filter Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	As selected in the MegaWizard interface.
Frame Height	As selected in the MegaWizard interface.
Interlaced/Progressive	Progressive.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color Pattern	One, two or three channels in sequence. For example, if three channels in sequence is selected, where a, b and g can be any color plane: 

2D Median Filter

The 2D Median Filter MegaCore function provides a means to perform 2D median filtering operations using matrices of 3×3, 5×5, or 7×7 kernels.

Each output pixel is the median of the input pixels found in a 3x3, 5x5, or 7×7 kernel centered on the corresponding input pixel. Where this kernel runs over the edge of the input image, zeros are filled in.

Larger kernel sizes require many more comparisons to perform the median filtering function and therefore require correspondingly large increases in the number of logic elements used. Larger sizes have a stronger effect, removing more noise but also potentially removing more detail.

All input data samples must be in unsigned format. If the number of bits per pixel per color plane is N, this means that each sample consists of N bits of data which are interpreted as an unsigned binary number in the range $[0, 2^N - 1]$. All output data samples produced by the 2D Median Filter MegaCore function are also in the same unsigned format.

The 2D Median Filter MegaCore function can process streams of pixel data of the types shown in Table 5–5.

Table 5–5. 2D Median Filter Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	As selected in the MegaWizard interface.
Frame Height	As selected in the MegaWizard interface.
Interlaced/Progressive	Progressive.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color Pattern	One, two or three channels in sequence. For example, if three channels in sequence is selected where a, b and g can be any color plane: 

Alpha Blending Mixer

The Alpha Blending Mixer MegaCore function provides an efficient means to mix together up to 12 image layers. The function provides support for both picture-in-picture mixing and image blending with per pixel alpha support.

The location and size of each layer can be changed dynamically while the MegaCore function is running, and individual layers can be switched on and off. This 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.

It is expected that each foreground layer fits in the boundaries of the background layer.

Control data is read in two steps at the start of each frame and is buffered inside the MegaCore function so that the control data can be updated during the frame processing without unexpected side effects.

The first step occurs after all the non-image data packets of the background layer have been processed and transmitted, and the core 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).

Non-image data packets of each active foreground layer, displayed or consumed, are processed in a sequential order, layer 1 first. Non-image data packets from the background layer are integrally transmitted whereas non-image data packets from the foreground layers are treated differently depending on their type. Control packets, of type 15, are processed by the core to extract the width and height of each layer and are discarded on the fly. Other packets, of type 1 to type 14, are propagated unchanged.

The second step corresponds to the usual behavior of other Video and Image Processing MegaCore functions that have an Avalon-MM slave interface.

After the non-image data packets from the background layer and the foreground layers have been processed and/or propagated, the MegaCore function waits for the `Go` bit to be set to 1 before reading the top left position of each layer.

Consequently, the behavior of the Alpha Blending Mixer differs slightly from the other Video and Image Processing MegaCore functions.

This behavior is 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();
}
```

When **Alpha blending** is on, the Avalon-ST input ports for the alpha channels expect a video stream compliant with the Avalon-ST Video protocol. Alpha frames contain a single color plane and are transmitted in video data packets. The first value in each packet, transmitted while the `startofpacket` signal is high, contains the packet type identifier 0. This holds true even when the width of the alpha channels data ports is less than 4-bits wide. The last alpha value for the bottom-right pixel is transmitted while the `endofpacket` signal is high.

It is not necessary to send control packets to the ports of the alpha channels. The width and height of each alpha layer are assumed to match with the dimensions of the corresponding foreground layer although the Alpha Blending Mixer MegaCore function recovers gracefully in the case of a mismatch. All non-image data packets (control packets included) are ignored and discarded just before the processing of a frame starts.

For information about using Avalon-MM slave interfaces for run-time control in the Video and Image Processing Suite, refer to "Avalon-MM Slave Interfaces" on page 4–14. For details of the register map for the Alpha Blending Mixer MegaCore function, refer to Table A–25 on page A–18.

The maximum number of image layers mixed cannot be changed dynamically and must be set in the MegaWizard interface for the Alpha Blending Mixer. The valid range of alpha coefficients is 0 to 1, where 1 represents full translucence, and 0 represents fully opaque.

For n-bit alpha values (RGBAn) 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.

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 $[0, 2^N - 1]$. All output data samples produced by the Alpha Blending Mixer MegaCore function are also in the same unsigned format.

The Alpha Blending Mixer MegaCore function can process streams of pixel data of the types shown in Table 5–6.

Table 5–6. Alpha Blending Mixer Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	Run time controlled. (Maximum value specified in the MegaWizard interface.)
Frame Height	Run time controlled. (Maximum value specified in the MegaWizard interface.)
Interlaced/Progressive	Progressive. Interlaced data is accepted but the offset from the top edge for each foreground layer has to be adapted in consequence.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface (specified separately for image data and alpha blending).
Color Pattern (din & dout)	One, two or three channels in sequence or in parallel as selected in the MegaWizard interface. For example, if three channels in sequence is selected where a, b and g can be any color plane: 
Color Pattern (alpha_in)	A single color plane representing the alpha value for each pixel: 

Scaler

The Scaler MegaCore function provides a means to resize video streams. It supports nearest neighbor, bilinear, bicubic, and polyphase scaling algorithms.

The Scaler MegaCore function can be configured to change the input resolution using control packets. It can also be configured to change the output resolution and/or filter coefficients at run time using an Avalon-MM Slave interface.

For information about using Avalon-MM slave interfaces for run-time control in the Video and Image Processing Suite, refer to "Avalon-MM Slave Interfaces" on page 4–14. For details of the register map for the Scaler MegaCore function, refer to Table A–26 on page A–19.

In the formal definitions of the scaling algorithms, the width and height of the input image are denoted w_{in} and h_{in} respectively. The width and height of the output image are denoted w_{out} and h_{out}. F is the function which returns an intensity value for a given point on the input image and O is the function which returns an intensity value on the output image.

Nearest Neighbor Algorithm

The nearest-neighbor algorithm used by the scaler is the lowest quality method, and uses the fewest resources. Jagged edges may be visible in the output image as no blending takes place. However, this algorithm requires no DSP blocks, and uses fewer logic elements than the other methods.

Scaling down requires no on-chip memory; scaling up requires one line buffer of the same size as one line from the clipped input image, taking account of the number of color planes being processed. For example, up scaling an image which is 100 pixels wide and uses 8-bit data with 3 colors in sequence but is clipped at 80 pixels wide, needs 8 × 3 × 80 = 1920 bits of memory. Similarly, if the 3 color planes are in parallel, the memory requirement is still 1920 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 $((i+0.5)\ w_{in}/w_{out},\ (j+0.5)\ h_{in}/h_{out})$.

The 0.5 values in this equation come from considering the coordinates of an image array to be on the lines of a 2D grid, but the pixels to be equally spaced between the grid lines that is, at half values.

This equation gives an answer relative to the mid-point of the input pixel and 0.5 should be subtracted to translate from pixel positions to grid positions. However, this 0.5 would then be added again so that later truncation performs rounding to the nearest integer. Therefore no change is needed. The calculation performed by the scaler is equivalent to the following integer calculation:

$$O(i, j) = F((2 \times w_{in} \times i + w_{in})/(2 \times w_{out}),\ (2 \times h_{in} \times j + h_{in})/(2 \times h_{out}))$$

Bilinear Algorithm

The bilinear algorithm used by the scaler is higher quality and more expensive than the nearest-neighbor algorithm. The jaggedness of the nearest-neighbor method is smoothed out, but at the expense of losing some sharpness on edges.

Resource Usage

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. Two line buffers are used. As in nearest-neighbor mode, each of line buffers is the size of a clipped line from the input image. The logic area used is more than that used by the nearest-neighbor method.

Algorithmic Description

This section describes how the algorithmic operations of the bilinear scaler can be modeled using a frame-based method. This does not reflect the implementation, but allows the calculations to be presented concisely. 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, B_{fh} and B_{fv}, respectively.

Their values can be calculated as:

$$err_i = \frac{((i \times w_{in})\%w_{out}) \times 2^{B_{fh}}}{max(w_{in}, w_{out})}$$

$$err_j = \frac{((j \times h_{in})\%h_{out}) \times 2^{B_{fv}}}{max(h_{in}, h_{out})}$$

where % is the modulus operator and $max(a, b)$ is a function that returns the maximum of two values.

The sum is then weighted proportionally to these errors. Note that because the 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:

$$O(i,j) \times 2^{B_{fv}+B_{fh}} = F(in_i, in_j) \times (2^{B_{fh}} - err_i) \times (2^{B_{fv}} - err_j) + F(in_i + 1, in_j) \times err_i \times (2^{B_{fv}} - err_j) + F(in_i, in_j + 1) \times (2^{B_{fh}} - err_i) \times err_j + F(in_i + 1, in_j + 1) \times err_i \times err_j$$

Polyphase and Bicubic Algorithms

The polyphase and bicubic algorithms offer the best image quality, but use more resources than the other modes of the scaler. They allow up 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 be used to reduce aliasing effects.

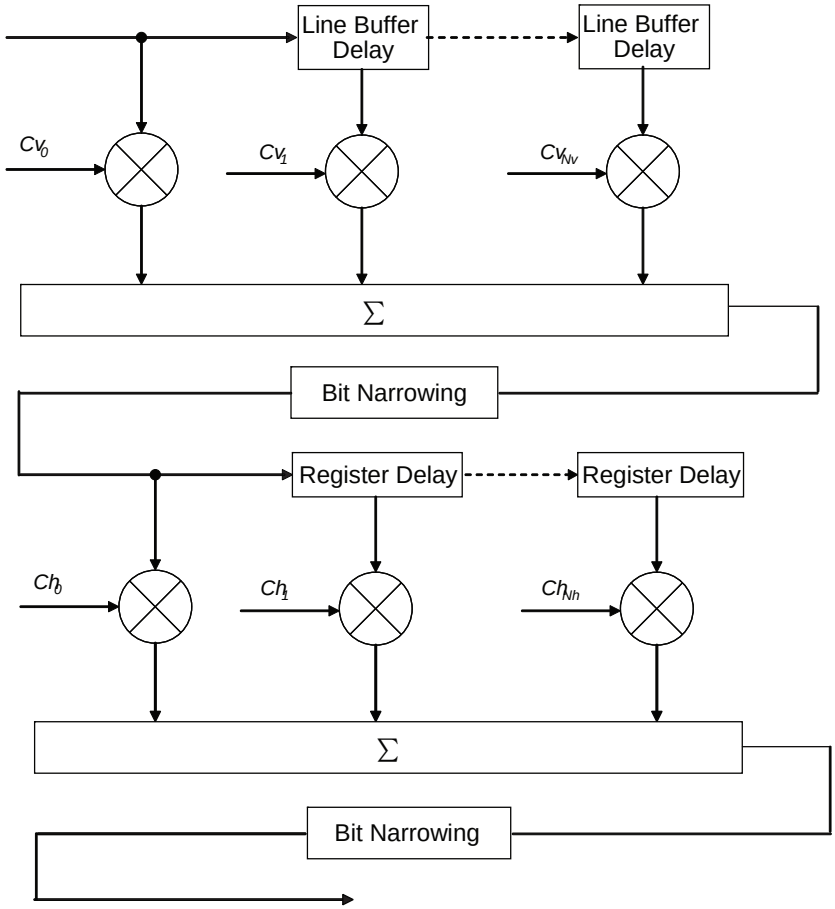
The bicubic and polyphase algorithms use different mathematics to derive their filter coefficients, but the implementation of the bicubic algorithm is just the polyphase algorithm with four vertical and four horizontal taps. In the following discussion, all comments relating to the polyphase algorithm are applicable to the bicubic algorithm assuming 4×4 taps.

Figure 5–4 on page 5–15 shows the flow of data through an instance of the scaler 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.

The progress of data through the taps (line buffer and register delays) and the coefficient values used in the multiplication are controlled by logic that is not present in the diagram. This logic is described in "Algorithmic Description" on page 5–16.

Figure 5–4. Polyphase Mode Scaler Block Diagram



Resource Usage

Consider an instance of the polyphase scaler with N_v vertical taps and N_h horizontal taps. B_{data} is the bit width of the data samples.

B_v is the bit width of the vertical coefficients and is derived from the user parameters for the vertical coefficients. It is equal to the sum of integer bits and fraction bits for the vertical coefficients, plus one if coefficients are signed.

B_h is defined similarly for horizontal coefficients. P_v and P_h are the user-defined number of vertical and horizontal phases for each coefficient set.

C_v is the number of vertical coefficient banks and C_h the 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. Refer to the **Number of bits to preserve between vertical and horizontal filtering** parameter in Table A–10 on page A–6.

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 Quartus II software maps these appropriately to physical on-chip RAM or logic elements as constrained by the width and depth requirements.

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 MegaWizard interface this setting is forced even when the coefficients are loaded at run time.

Using multiple coefficient banks allows double-buffering, fast swapping, or direct writing to the Scaler's coefficient memories. The coefficient bank to be read during video data processing and the bank to be written by the Avalon-MM interface are specified separately at runtime (Refer to the control register map in Table A–26 on page A–19). This means that you can accomplish double-buffering by performing the following steps:

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, and start processing.
4. For subsequent changes, write to the unused bank (1) and swap the read/write banks between frames.

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, a single bank can be used and coefficient writes would have an immediate effect on data processing.

Algorithmic Description

This section describes how the algorithmic operations of the polyphase scaler can be modelled using a frame-based method. This description shows how the filter kernel is applied and how coefficients are loaded, but is not intended to indicate how the hardware of the scaler is designed.

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 should be at the mid-point of the pixel to output. This is achieved be 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 $(\lfloor in_i \rfloor, \lfloor in_j \rfloor)$ where in_i and in_j are calculated using the algorithmic description equations on page 5–13. The difference between the real and integer solutions to these equations determines the position of the filter function used during scaling.

The filter function is positioned over the real solution by adjusting the function's phase:

$$phase_i = \frac{((i \times w_{in})\%w_{out}) \times P_h}{max(w_{in}, w_{out})}$$

$$phase_j = \frac{((j \times h_{in})\%h_{out}) \times P_v}{max(h_{in}, h_{out})}$$

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.

Choosing and Loading Coefficients

The filter coefficients used by the polyphase mode of the scaler may be specified at compile time or at run time. At compile time, the coefficients can be either selected from a set of Lanczos-windowed sinc functions, or loaded from a comma-separated variable (CSV) file.

At run time they are specified by writing to the Avalon-MM slave control port (Table A–26 on page A–19).

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 MegaCore function processes active data without causing corruption.

Figure 5–5 on page 5–18 shows how a 2-lobe Lanczos-windowed sinc function (usually referred to as Lanczos 2) would be sampled for a 4-tap vertical filter.

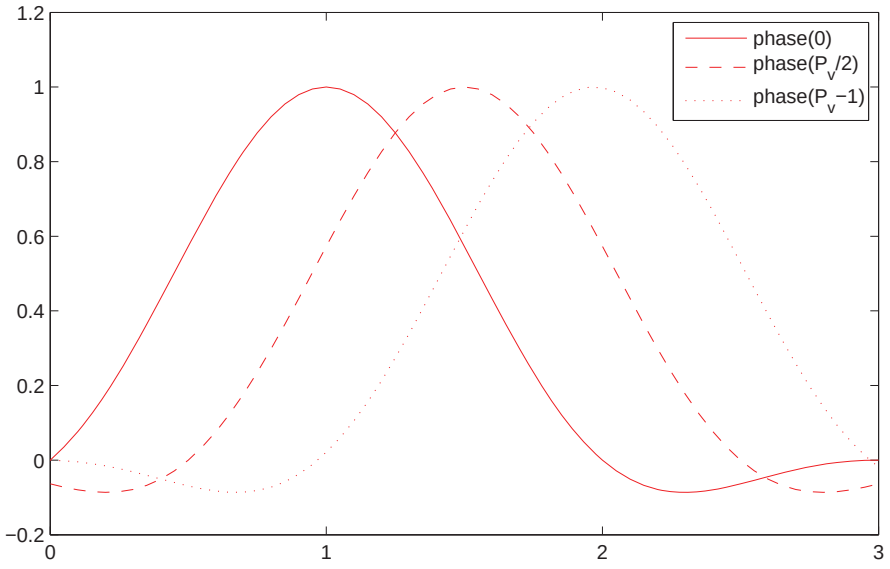
The two lobes refer to the number of times the function changes direction on each side of the central maxima, including the maxima itself.

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 centres 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.

Figure 5–5. Lanczos 2 Function at Various Phases



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 $N \times P$ values. The values must be listed as N taps in order for phase 0, N taps for phase 1, up to the Nth tap of the Pth phase. Values do not need to be presented 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 MegaWizard interface, 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 should be in the range [–256, 255] representing the range [-2, 1.9921875].

In summary, you can generate a set of coefficients for an N-tap, P-phase instance of the Scaler as follows:

1. Define a function, $f(x)$ over the domain $[0, N-1]$ under the assumption that $(N-1)/2$ is the mid-point of the filter.
2. For each tap $t \in \{0, 1, \ldots, N-1\}$ and for each phase $p \in \{0, 1/P, \ldots, (P-1/P)\}$, sample $f(t-p)$.
3. Quantize each sample. Ideally, the sum of the quantized values for all phases should be equal.
4. Either store these in a CSV file and copy them into the MegaWizard interface, or load them at run time using the control interface.

Coefficients for the bicubic algorithm are calculated using Catmull-Rom splines to interpolate between values in tap 1 and tap 2.

For detailed information about the mathematics used for Catmull-Rom splines refer to *E Catmull and R Rom. A class of local interpolating splines. Computer Aided Geometric Design, pages 317–326, 1974.*

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 Œ $[1, P-1]$ and for all t Œ $[0, N-1]$, $C[p][t] = C[P-p][N-1-t]$

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. Selecting **Symmetric** for a coefficients set on the **Coefficients** page in the MegaWizard interface, forces the coefficients to be symmetric.

Recommended Parameters

In polyphase mode, you must choose parameters for the Scaler Megacore function carefully to get the best image quality.

Incorrect parameters can cause a decrease in image quality even as the resource usage increases. The parameters which have the largest effect are the number of taps and the filter function chosen to provide the coefficients. The number of phases and number of bits of precision used are less important to the image quality.

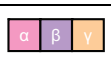
Table 5–7 summarizes some recommended values for parameters when using the Scaler in polyphase mode.

Table 5–7. Recommended Parameters for the Scaler Megacore Function

Scaling Problem	Taps	Phases	Precision	Coefficients
Scaling up with any input/output resolution	4	16	Signed, 1 integer bit, 7 fraction bits	Lanczos-2, or Bicubic
Scaling down from *M* pixels to *N* pixels	$\frac{M \times 4}{N}$	16	Signed, 1 integer bit, 7 fraction bits	Lanczos-2
Scaling down from *M* pixels to *N* pixels (lower quality)	$\frac{M \times 2}{N}$	16	Signed, 1 integer bit, 7 fraction bits	Lanczos-1

The Scaler MegaCore function can process streams of pixel data of the types shown in Table 5–8.

Table 5–8. Scaler Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	Maximum frame width is specified in the MegaWizard interface, the actual value is read from control packets.
Frame Height	Maximum frame height is specified in the MegaWizard interface, the actual value is read from control packets.
Interlaced/Progressive	Progressive.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color Pattern	One, two or three channels in sequence or in parallel as selected in the MegaWizard interface. For example, if three channels in sequence is selected where a, b and g can be any color plane: 

Clipper

The Clipper MegaCore function provides a means to select an active area from a video stream and discard the remainder.

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

The Clipper can deal with changing input resolutions by reading Avalon-ST Video control packets. An optional Avalon-MM interface allows the clipping settings to be changed at runtime.

The Clipper MegaCore function can process streams of pixel data of the types shown in Table 5–9.

Table 5–9. Clipper Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	Maximum frame width is specified in the MegaWizard interface, the actual value is read from control packets.
Frame Height	Maximum frame height is specified in the MegaWizard interface, the actual value is read from control packets.
Interlaced/Progressive	Either.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color Pattern	One, two or three channels in sequence or in parallel as selected in the MegaWizard interface. For example, if three channels in sequence is selected where a, b and g can be any color plane: 

Deinterlacer

The Deinterlacer MegaCore function converts interlaced video to progressive video using bob, weave, or motion-adaptive methods. In addition, the Deinterlacer can provide double or triple-buffering in external RAM. Buffering is required by the motion-adaptive and weave methods and can be selected if desired when using a bob method.

You can configure the Deinterlacer to produce one output frame for each input field or to produce one output frame for each input frame (a pair of two fields). For example, if the input video stream is NTSC video with 60 interlaced fields per second, the former configuration outputs 60 frames per second but the latter outputs 30 frames per second.

When you select a frame buffering mode, the Deinterlacer output is calculated in terms of the current field and possibly some preceding fields. For example, the bob algorithm uses the current field, whereas the weave algorithm uses both the current field and the one which was received immediately before it. When producing one output frame for every input field, each field in the input frame takes a turn at being the current field.

However, if one output frame is generated for each pair of interlaced fields then the current field moves two fields through the input stream for each output frame. This means that the current field is either always a F0 field (defined as a field which contains the top line of the frame) or always a F1 field. The Deinterlacer MegaCore function allows the choice of F0 or F1 to be made at compile time.

Deinterlacing Methods

The Deinterlacer MegaCore function supports four deinterlacing methods:

- Bob with scanline duplication
- Bob with scanline interpolation
- Weave
- Motion-adaptive

Bob with Scanline Duplication

The bob with scanline duplication algorithm is the simplest and cheapest in terms of logic. Output frames are produced by simply repeating every line in the current field twice. Because only the current field is used, if the output frame rate is selected to be the same as the input frame rate then half of the input fields are discarded.

Bob with Scanline Interpolation

The bob with scanline interpolation algorithm has a slightly higher logic cost than bob with scanline duplication but offers significantly better quality.

Output frames are produced by filling in the missing lines from the current field with the linear interpolation of the lines above and below them. 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. Because only the current field is used, if the output frame rate is selected to be the same as the input frame rate then half of the input fields are discarded.

Weave

Weave deinterlacing creates an output frame by filling all of the missing lines in the current field with lines from the previous field. This option gives good results for still parts of an image but unpleasant artefacts in moving parts.

The weave algorithm requires external memory, so either double or triple-buffering must be selected. This makes it significantly more expensive in logic elements and external RAM bandwidth than either of the bob algorithms, if external buffering is not otherwise required.

The results of the weave algorithm can sometimes be perfect, in the instance where pairs of interlaced fields have been created from original progressive frames. Weave simply stitches the frames back together and the results are the same as the original, as long as output frame rate equal to input frame rate is selected and the correct pairs of fields are put together. Usually progressive sources split each frame into a pair consisting of an F0 field followed by an F1 field, so selecting F1 to be the current field often yields the best results.

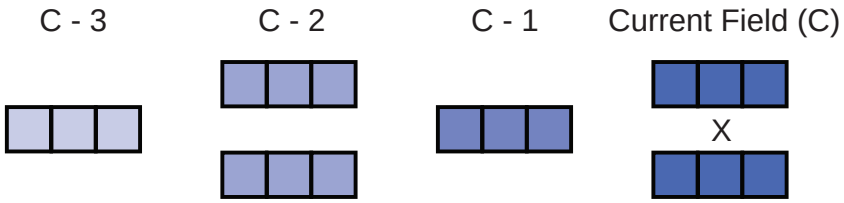
Motion-Adaptive

The Deinterlacer MegaCore function provides a simple motion-adaptive algorithm. This is the most sophisticated of the algorithms provided but also the most expensive, both in terms of logic area and external memory bandwidth requirement.

This algorithm avoids the weaknesses of bob and weave algorithms by using a form of bob deinterlacing for moving areas of the image and weave style deinterlacing for still areas. To achieve this, the 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:

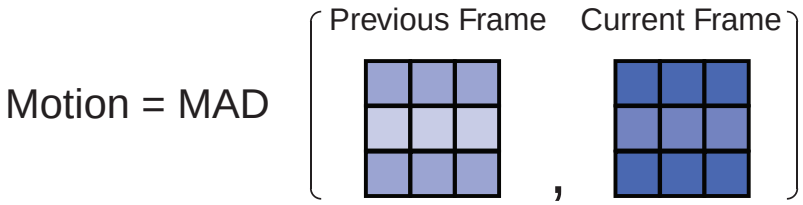
1. Pixels are collected from the current field and the three preceding it (the *X* denotes the location of the desired output pixel) as shown in Figure 5–6.

Figure 5–6. Pixel Collection for the Motion-Adaptive Algorithm



2. These pixels are assembled into two 3×3 groups of pixels and the minimum absolute difference of the two groups as shown in Figure 5–7.

Figure 5–7. Pixel Assembly for the Motion-Adaptive Algorithm



3. The minimum absolute difference value is normalized into the same range as the input pixel data. If the **Motion bleed** algorithm is selected, the motion value is compared with a recorded motion value for the same location in the previous frame. If it is greater, the new value is kept, but if the new value is less than the stored value, the motion value used is the mean of the two values. This reduces unpleasant flickering artefacts but increases the memory usage and memory bandwidth requirements.
4. Two pixels are selected for interpolation by examining the 3×3 group of pixels from the more recent two fields for edges. If a diagonal edge is detected, the two pixels from the current field that lie on the diagonal are selected, otherwise the pixels directly above and below the output pixel are chosen.
5. The output pixel is calculated using a weighted mean of the interpolation pixels and the equivalent to the output pixel in the previous field as follows:

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

The motion-adaptive algorithm requires the buffering of two frames of data before it can produce any output. For this reason, the Deinterlacer always consumes the three first fields it receives at start up and after a change of resolution without producing any output.

Pass-Through Mode for Progressive Frames

In its default configuration, the Deinterlacer discards progressive frames. This behavior can be changed if you want a data path compatible with both progressive and interlaced inputs and where run-time switching between the two types of input is allowed. When the deinterlacer is configured to let progressive frames pass through, the deinterlacing algorithm in use (bob, weave or motion-adaptive) propagates progressive frames unchanged. The double/triple-buffering function is maintained while propagating progressive frames.

Enabling the propagation of progressive frames has an impact on memory usage in all the parameterizations of the bob algorithm that use buffering. The motion-adaptive algorithm writes and reads motion values from the memory uninterruptedly, even when progressive frames are passed on unchanged.

Frame Buffering

The Deinterlacer MegaCore function also allows frame buffering in external RAM to be configured at compile time. When using either of the two bob algorithm subtypes, no buffering, double-buffering, or triple-buffering can be selected. The weave and motion-adaptive algorithms require some external frame buffering, and in those cases only double-buffering or triple-buffering can be selected.

When no buffering is chosen, input pixels flow into the Deinterlacer through its input port and, after some delay, calculated output pixels flow out through the output port. When double-buffering is selected, two frame buffers are used in external RAM. Input pixels flow through the input port and into one buffer while pixels are read from the other buffer, processed and output.

When both the input and output sides have finished processing a frame, the buffers swap roles so that the frame which has just been input can be used by the output. The frame which has just been used to create output can then be overwritten with fresh input.

The motion-adaptive algorithm uses four fields to build a progressive output frame and the output side has to read pixels from two frame buffers rather than one. Consequently, the motion-adaptive algorithm actually uses three frame buffers in external RAM when double-buffering is selected. When the input and output sides have finished processing a frame, the output side exchanges its buffer containing the oldest frame, frame n-2, with the frame just received at the input side, frame n, but it keeps frame n-1 for one extra iteration because it is used along with frame n to produce the next output.

When triple-buffering is in use, three frame buffers are usually used in external RAM. Four frame buffers are used when the motion-adaptive algorithm is selected for the reasons described in the previous paragraph. At any time, one buffer is in use by the input and one (two for the motion adaptive case) is (are) in use by the output in the same way as the double-buffering case. The last frame buffer is spare.

This configuration allows the input and output sides to swap asynchronously. When the input has finished, it swaps with the spare frame if the spare frame contains data which has already been used by the output frame. Otherwise the frame which has just been written is dropped and a fresh frame is written over the dropped frame.

When the output has finished, it also swaps with the spare frame and continues provided that the spare frame contains fresh data from the input side, otherwise it does not swap and just repeats the last frame.

Triple-buffering therefore allows simple frame rate conversion. For example, suppose the Deinterlacer's input is connected to a HDTV video stream in 1080i60 format and its output is connected to a 1080p50 monitor. The input has 60 interlaced fields per second, but the output tries to pull 50 progressive frames per second.

If you configure the Deinterlacer to output one frame for each input field, it produces 60 frames of output per second. If triple-buffering is enabled, then on average one frame in six is dropped so that 50 frames per second are produced. If the chosen configuration is one frame output for every pair of fields input, the Deinterlacer produces 30 frames per second output and triple-buffering leads to two out of every three frames being repeated on average.

When double or triple-buffering is selected the Deinterlacer has two or more Avalon-MM master ports. These must be connected to an external memory with enough space for all of the frame buffers required. The amount of space varies depending on the type of buffering and algorithm selected. An estimate of the required memory is shown in the Deinterlacer MegaWizard interface.

If the external memory in your system runs at a different clock rate to the Deinterlacer MegaCore function, you can turn on an option to use a separate clock for the Avalon-MM master interfaces and use the memory clock to drive these interfaces.

Frame Rate Conversion

When triple-buffering is selected, the decision to drop and repeat frames is based on the status of the spare buffer. Because the input and output sides are not tightly synchronized, the behavior of the Deinterlacer 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. By using a double-buffer and controlling the dropping/repeating behavior, the input and output can be kept synchronized. For example, if the input has 60 interlaced fields per second, but the output requires 50 progressive frames per second (fps), setting the input frame rate to 30 fps and the output frame rate at 50 fps guarantees that exactly one frame in six is dropped.

To control the dropping/repeating behavior and to synchronize the input and output sides, you must select double-buffering mode and turn on **Runtime control for locked frame rate conversion** in the Parameter Settings page of the MegaWizard interface. The input and output rates can be selected and changed at run time. Table A–30 on page A–22 describes the control register map.

The rate conversion algorithm is fully compatible with a progressive input stream when the progressive passthrough mode is enabled but it cannot be enabled simultaneously with the run-time override of the motion-adaptive algorithm.

Behavior When Unexpected Fields are Received

So far, the behavior of the Deinterlacer has been described assuming an uninterrupted sequence of pairs of interlaced fields (F0, F1, F0, …). Some video streams might not follow this rule and the Deinterlacer adapts its behavior in such cases.

The dimensions and type of a field (progressive, interlaced F0, or interlaced F1) are identified using information contained in Avalon-ST Video control packets. When a field is received without control packets, its type is defined by the type of the previous field. A field following a progressive field is assumed to be a progressive field and a field following an interlaced F0 or F1 field is respectively assumed to be an interlaced F1 or F0 field. If the first field received after reset is not preceded by a control packet, it is assumed to be an interlaced field and the default initial field (F0 or F1) specified in the MegaWizard interface is used.

When the weave or the motion-adaptive algorithms are used, a regular sequence of pairs of fields is expected. Subsequent F0 fields received after an initial F0 field or subsequent F1 fields received after an initial F1 field are immediately discarded.

When the bob algorithm is used and synchronization is done on a specific field (input frame rate = output frame rate), the field that is constantly unused is always discarded. The other field is used to build a progressive frame, unless it is dropped by the triple-buffering algorithm.

When the bob algorithm is used and synchronization is done on both fields (input field rate = output frame rate), the behavior is dependent on whether buffering is used. If double or triple-buffering is used, the bob algorithm behaves like the weave and motion-adaptive algorithms and a strict sequence of F0 and F1 fields is expected. If two or more fields of the same type are received successively, the extra fields are dropped. When buffering is not used, the bob algorithm always builds an output frame for each interlaced input field received regardless of its type.

If pass- through mode for progressive frames has not been selected, the Deinterlacer immediately discards progressive fields in all its parameterizations.

Handling of Avalon-ST Video Control Packets

When buffering is used, the Deinterlacer MegaCore function stores non-image data packets in memory as described in "Buffering of Non-Image Data Packets in Memory" on page 4–17.

Control packets and user packets are never repeated and they are not dropped as long as memory space is sufficient. In all parameterizations, including those that do not use buffering in external memory, the Deinterlacer MegaCore function does not discard nor modify the control packets associated with the incoming interlaced fields. However, an additional control packet containing the correct frame height and the proper interface flag for progressive fields is inserted just before the frame so that the image data packet is interpreted correctly by following MegaCore functions.

The Deinterlacer MegaCore function can process streams of pixel data of the types shown in Table 5–10.

Table 5–10. Deinterlacer Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	Run time controlled. (Maximum value specified in the MegaWizard interface.)
Frame Height	Run time controlled. (Maximum value specified in the MegaWizard interface.)
Interlaced/Progressive	Interlaced input, Progressive output.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color Pattern	One, two or three channels in sequence or in parallel as selected in the MegaWizard interface. For example, if three channels in sequence is selected where a, b and g can be any color plane: 

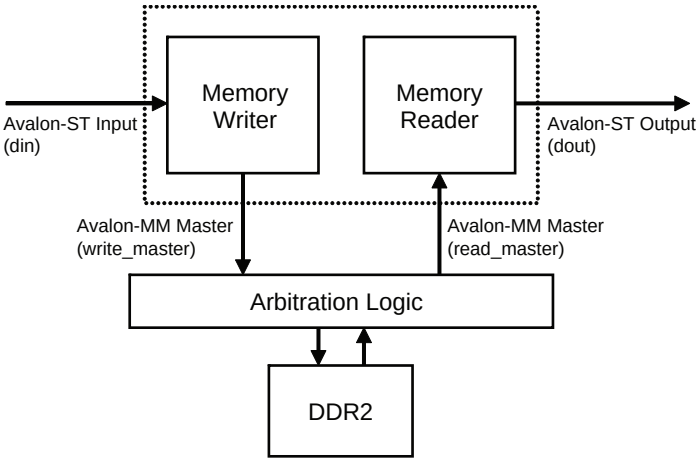
Frame Buffer

The Frame Buffer MegaCore function buffers video frames in external RAM. When frame dropping and frame repeating are not allowed, the Frame Buffer provides a double-buffering function that can be useful to solve throughput issues in the data path. When frame dropping and/or frame repeating are allowed, the Frame Buffer provides a triple-buffering function and can be used to perform simple frame rate conversion.

The Frame Buffer is built with two basic blocks: a writer which stores input pixels in memory and a reader which retrieves video frames from the memory and outputs them.

Figure 5–8 shows a simple block diagram of the Frame Buffer Megacore function.

Figure 5–8. Frame Buffer Block Diagram



When double-buffering is in use, two frame buffers are used in external RAM. At any time, one buffer is used by the writer component to store input pixels, while the second buffer is locked by the reader component that reads the output pixels from the memory.

When both the writer and the reader components have finished processing a frame, the buffers are exchanged. The frame that has just been input 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.

A double-buffer is typically 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 double-buffer is often used when 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 MegaCore function or before one of the foreground layers of the Alpha Blending Mixer MegaCore function.

When triple-buffering is in use, three frame buffers are used in external RAM. As was the case in double-buffering, the reader and the writer components are always locking one buffer to respectively store input pixels to memory and read output pixels from memory. The third frame buffer is a spare buffer that allows the input and the output sides to swap buffers asynchronously. The spare buffer is considered *clean* if it contains a fresh frame that has not been output, or *dirty* if it contains an old frame that has already been sent by the reader component.

When the writer has finished 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 component becomes the new spare buffer and is *clean* because it contains a fresh frame. If the spare buffer is already *clean* when the writer has finished writing the current input frame and if dropping frames is allowed, then the writer drops the frame that has just been received and overwrites its buffer with the next incoming frame. If dropping frames is not allowed, the writer component stalls until the reader component has finished its frame and replaced the spare buffer with a *dirty* buffer.

Similarly, when the reader has finished reading and has output a frame from memory, it swaps its buffer with the spare buffer if the spare buffer is *clean*. The buffer locked by the reader component 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 has finished the current output frame and if repeating frames are allowed, the reader immediately repeats the frame that has just been sent.

If repeating frames are not allowed, the reader component stalls until the writer component has finished its frame and replaced the spare buffer with a "clean" buffer.

Triple-buffering therefore allows simple frame rate conversion to be performed when the input and the output are pushing and pulling frames at different rates.

Handling of Avalon-ST Video Control Packets

The Frame Buffer MegaCore function stores non-image data packets in memory as described in "Buffering of Non-Image Data Packets in Memory" on page 4–17. Control packets and user packets are never repeated and they are not dropped as long as the memory space is sufficient.

When a frame is dropped by the writer, the non-image data packets that preceded it are kept and sent with the next frame that is not dropped. When a frame is repeated by the reader, it is repeated without the packets that preceded it.

The Frame Buffer MegaCore function propagates Avalon-ST Video control packets without reading them and relies exclusively on the `startofpacket` and `endofpacket` signals to delimit the frame boundaries. The Frame Buffer can consequently handle mislabelled frames seamlessly, but this comes with a small cost. The Frame Buffer cannot predict when the endofpacket signal is received, and if the dimensions of a field do not match the maximum width and height specified at compile time in the MegaWizard interface, some extra unused data may be written in memory to conclude a transaction (at most one line of data).

The Frame Buffer does not differentiate between interlaced and progressive fields. When interlaced fields are received, the Frame Buffer buffers, drops or repeats fields independently, and support for the interlaced video stream is consequently incomplete.

The Frame Buffer MegaCore function can process streams of pixel data of the type shown in Table 5–11.

Table 5–11. Frame Buffer Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	Run time controlled. Maximum value selected in the MegaWizard interface.
Frame Height	Run time controlled. Maximum value selected in the MegaWizard interface.
Interlaced/Progressive	Progressive, although interlaced data can be accepted in some cases.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color Pattern	Any combination of one, two or three channels in each of sequence or parallel. For example, if three channels in sequence is selected where a, b and g can be any color plane: 

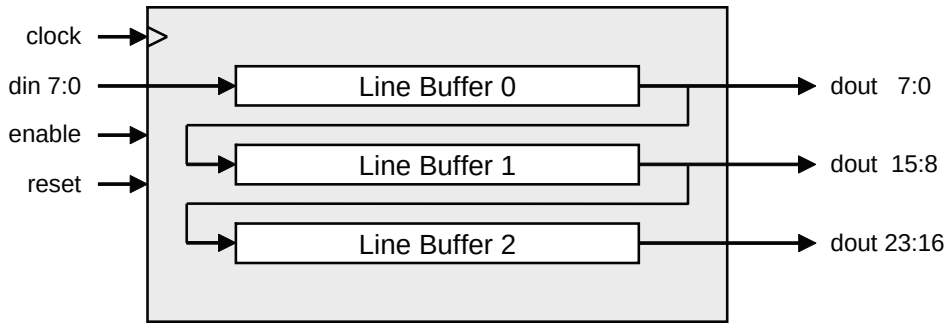
Line Buffer Compiler

FPGA memory is a valuable resource for many video and imaging applications, particularly when developing systems that require high-definition resolutions and high-order-accuracy algorithms.

The Line Buffer Compiler provides an efficient means to map line buffers to Altera on-chip memories.

An example of the logic structure produced by the Line Buffer Compiler is shown in Figure 5–9.

Figure 5–9. Example of the Line Buffer Compiler Logic



In this example, there are three line buffers, each of which is eight bits wide. It is possible to use the Line Buffer Compiler MegaCore function to create similar structures with up to sixteen line buffers, each up to 64 bits in width.

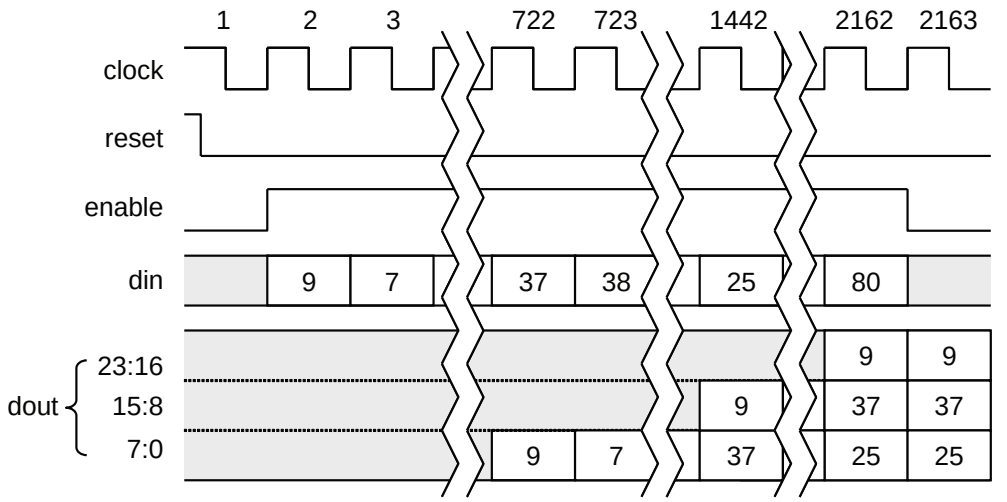
When `enable` is asserted, data flows into the module through the `din` port and passes through each of the line buffers in sequence.

The output of all of the line buffers is concatenated together into a bus of size (*number of line buffers*) × (*line buffer width*) bits which can be read at any time.

Unlike the other MegaCore functions in the Video and Image Processing Suite, the Line Buffer Compiler does not provide an Avalon-ST Video protocol based data interface.

Figure 5–10 shows a timing diagram illustrating how a Line Buffer Compiler MegaCore function such as the one shown in Figure 5–9 would process data if the length of each line was set to 720 pixels.

Figure 5–10. Timing diagram Illustrating a Line Buffer Compiler in Use



The sequence of events (at 1, 2, 3, 722, 723 1442, 2162 and 2163 cycles) is shown in Figure 5–10 is as follows:

1. `reset` is deasserted synchronous to `clock` and the MegaCore function becomes ready for use. The contents of all of the line buffers is undefined, as is the state of the output, `dout`.
2. `enable` is driven high and the value `9` is driven onto the input bus `din`. This value is captured and stored in the first location of the first line buffer.
3. `enable` stays high and the value `7` is driven onto the input bus. All of the data in all of the line buffers (currently just the `9`) moves along one place and `7` moves into the first location of the first line buffer.

722. 720 enabled clock cycles after the value `9` was captured on the input bus `din`, the value `9` is driven on to the output of the first line buffer. This is connected to the bottom eight bits of `dout`. The number `37` is driven on to the input.

723. The second input value, a 7, reaches the end of the first line buffer and is visible in the bottom eight bits of dout. The value 9 is now in the first location in the second line buffer.

1442. 1440 enabled clock cycles after the value 9 was captured on the input it reaches the end of the second line buffer and is output on the middle eight bits of dout. 37 has reached the end of the first line buffer and is driven on to the low eight bits. The value 25 is driven on to the input.

2162. 2160 enabled clock cycles after the value 9 was captured it reaches the end of the last line buffer and is driven on to the top eight bits of dout. The middle and bottom sets of eight bits of dout show the data words captured 720 and 1440 enabled clock cycles after it, respectively.

2163. When enable is deasserted, any input value on din is not captured and the contents of the line buffers remains unchanged. It is still possible to read the same output values from dout.

Clocked Video Input

The Clocked Video Input MegaCore function converts from clocked video formats (such as BT656 and DVI) to Avalon-ST Video.

The Clocked Video Input strips the incoming clocked video of horizontal and vertical blanking, leaving only active picture data, and using this data with the horizontal and vertical sync information creates the necessary Avalon-ST Video control and active picture packets. No conversion is done to the active picture data, the color plane information remains the same as in the clocked video format.

The Clocked Video Input converts clocked video to the flow controlled Avalon-ST Video protocol. It also provides clock crossing capabilities to allow video formats running at different frequencies to enter the system.

In addition, the Clocked Video Input provides a number of status registers that provide feedback on the format of video entering the system (resolution, and interlaced or progressive mode) and a status interrupt that can be used to determine when the video format changes or is disconnected.

Video Formats

The Clocked Video Input MegaCore function accepts the following clocked video formats:

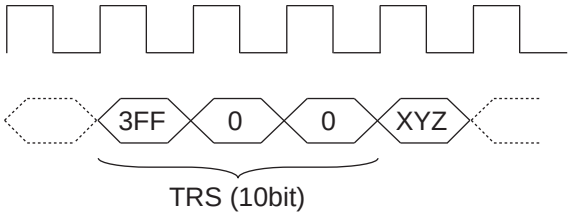
- Video with sync information embedded in the data (in BT656 format)
- Video with separate sync (H sync, Vsync) signals

Embedded Sync Format

BT656 uses time reference signal (TRS) codes in the video data to mark the places where sync information is inserted in the data.

These codes are made up of values that are not present in the video portion of the data and take the format shown in Figure 5–11.

Figure 5–11. Time Reference Signal Format



A BT656 interface can use 8 bits or 10 bits of multiplexed Y'CbCr data. To convert from the 10-bit to the 8-bit version, remove the two LSBs.

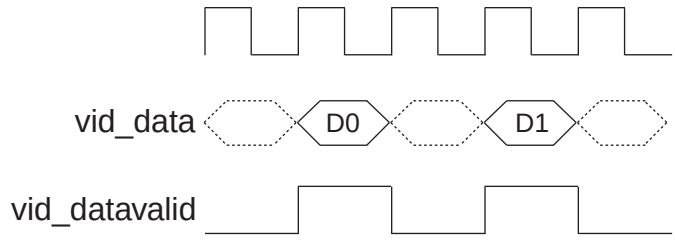
The XYZ word contains the sync information and the relevant bits of it's format are shown in Table 5–12.

Table 5–12. XYZ Word Format

	10-bit	8-bit	Description
Unused	[5:0]	[3:0]	These bits are not inspected by the Clocked Video Input MegaCore function.
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 Clocked Video Input MegaCore function.

For the embedded sync format, the `vid_datavalid` signal is used to indicate a valid BT656 sample as shown in Figure 5–12. The Clocked Video Input MegaCore function only reads the `vid_data` signal when `vid_datavalid` is 1.

Figure 5–12. vid_datavalid Timing



Separate Sync Format

The separate sync format uses separate signals to indicate the blanking, sync, and field information. For this format, the `vid_datavalid` signal behaves slightly differently from in embedded sync format.

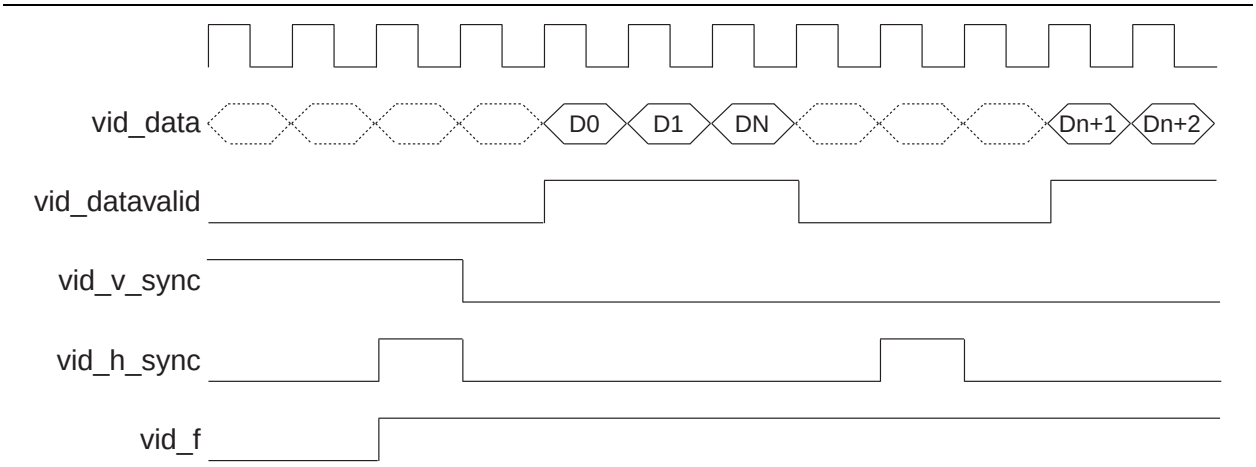
The Clocked Video Input MegaCore function only reads `vid_data` when `vid_datavalid` is high (as in the embedded sync format) but it treats each read sample as active picture data.

Table 5–13 describes the signals and Figure 5–13 shows the timing.

Table 5–13. Clocked Video Input Signals for Separate Sync Format Video

Signal Name	Description
`vid_datavalid`	When asserted the video is in an active picture period (not horizontal or vertical blanking).
`vid_h_sync`	When 1, the video is in a horizontal sync period.
`vid_v_sync`	When 1, the video is in a vertical sync 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.

Figure 5–13. Separate Sync Signals Timing



Video Locked Signal

The `vid_locked` signal indicates that the clocked video stream is active. When the signal has a value of 1, the Clocked Video Input MegaCore function takes the input clocked video signals as valid and reads and processes 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) the Clocked Video Input MegaCore function takes the input clocked video signals as invalid and does not process them.

If the `vid_locked` signal goes invalid while a frame of video is being processed, the Clocked Video Input MegaCore function ends the frame of video early.

Control Port

If you turn on **Use control port** in the MegaWizard interface for the Clocked Video Input, its Avalon-ST Video output can be controlled using the Avalon-MM slave control port.

Initially, the MegaCore function is disabled and does not output any data. However, it still detects the format of the clocked video input and raises interrupts.

The sequence for starting the output of the MegaCore function is as follows:

1. Write a 1 to `Control` register bit 0.
2. Read `Status` register bit 0. When this is a 1, the MegaCore function outputs data. This occurs on the next start of frame or field that matches the setting of the **Field order** in the MegaWizard interface.

The sequence for stopping the output of the MegaCore function is as follows:

1. Write a 0 to `Control` register bit 0.
2. Read `Status` register bit 0. When this is a 0, the MegaCore function has stopped data output. This occurs on the next end of frame or field that matches the setting of the **Field order** in the MegaWizard interface.

The starting and stopping of the MegaCore function is synchronized to a frame or field boundary.

Table 5–14 shows the output of the MegaCore function with the different **Field order** settings.

Table 5–14. Sync Settings

Video Format	Field Order	Output
Interlaced	F1 first	Start, F0, F1, ..., F0, F1, Stop
Interlaced	F0 first	Start, F1, F0, ..., F1, F0, 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

Format Detection

The Clocked Video Input MegaCore function detects the format of the incoming clocked video and uses it to create the Avalon-ST Video control packet. It also provides this information in a set of registers.

The MegaCore function can detect the following different aspects of the incoming video stream:

- Active picture width (in samples)—The MegaCore function counts the number of samples per line in the active picture period. The MegaCore function requires one full line of video before it can determine the width.
- Active picture height (in lines)—The MegaCore function counts the number of lines per frame or field in the active picture period. The MegaCore function requires one full frame or field of video before it can determine the height.
- Interlaced/Progressive—The MegaCore function detects whether the incoming video is interlaced or progressive. If it is interlaced, the MegaCore function stores separate width and height values for both fields. The MegaCore function requires two full frames or fields of video before it can determine whether the source is interlaced or progressive.

If the MegaCore function has not yet determined the format of the incoming video, it uses the values specified in the **Avalon-ST Video Initial/Default Control Packet** section of the MegaWizard interface.

After determining an aspect of the incoming videos format, the MegaCore function enters the value in the respective register, sets the registers valid bit in the `Status` register, and triggers the status update interrupt.

Table 5–15 shows the sequence for a 1080i incoming video stream.

Table 5–15. Resolution Detection Sequence for a 1080i Incoming Video Stream

Status	F0SampleCount	F0LineCount	F1SampleCount	F1LineCount	Description
0000000000	0	0	0	0	Start of incoming video.
0000001010	1,920	0	0	0	F0 - after one line of active picture (interrupt triggers).
0000011010	1,920	1,080	0	0	F0 - after one field of active picture (interrupt triggers).
0000111010	1,920	1,080	1,920	0	F1 - after one line of active picture (interrupt triggers).
0011111010	1,920	1,080	1,920	1,080	F1 - after one field of active picture (interrupt triggers).
0111111110	1,920	1,080	1,920	1,080	After 2 full frames of active picture the stable bit is set (interrupt triggers).

Interrupt

The Clocked Video Input MegaCore function outputs a single interrupt line which is the `OR` of the following internal interrupts:

- The status update interrupt—Triggers when the `Status` register is updated due to a detection of an aspect of the incoming video stream. The interrupt also triggers if it detects a loss of the `vid_locked` signal (if, for example, the video cable is removed). In this case, all the valid bits in the `Status` register are set to 0.
- Stable video interrupt—Triggers when the incoming video has had a consistent format for two frames.

Both interrupts can be independently enabled using bits [2:1] of the `Control` register. Their values can be read using bits [2:1] of the `Status` register and a write of 1 to either of those bits clears the respective interrupt.

Overflow

Moving between the domain of clocked video and the flow controlled world of Avalon-ST Video can cause problems if the flow controlled world does not accept data at a rate fast enough to satisfy the demands of the incoming clocked video.

The Clocked Video Input MegaCore function contains a FIFO that, when set to a large enough value, can accommodate any bursts in the flow data, as long as the input rate of the upstream Avalon-ST Video components is equal to or higher than that of the incoming clocked video.

If this is not the case, the FIFO overflows. If overflow occurs, the MegaCore function outputs an early endofpacket signal to complete the current frame. It then waits for the next start of frame (or field) before re-synchronizing to the incoming clocked video and beginning to output data again.

The overflow is recorded in bit [9] of the `Status` register. This bit is sticky, and if an overflow occurs, stays at 1 until the bit is cleared by writing a 0 to it.

In addition to the overflow bit, the current level of the FIFO can be read from the `UsedW` register.

Timing Constraints

To constrain the Clocked Video Output MegaCore function correctly, add the following file to your Quartus II project:

<*install_dir*>\ip\clocked_video_input\lib\alt_vip_cvi.sdc

Clocked Video Output

The Clocked Video Output MegaCore function converts Avalon-ST Video to clocked video formats (such as BT656 and DVI). It formats Avalon-ST Video into clocked video by inserting horizontal and vertical blanking and generating horizontal and vertical sync information using the Avalon-ST Video control and active picture packets.

No conversion is done to the active picture data, the color plane information remains the same as in the Avalon-ST Video format.

The Clocked Video Output MegaCore function converts data from the flow controlled Avalon-ST Video protocol to clocked video. It also provides clock crossing capabilities to allow video formats running at different frequencies to be output from the system.

In addition, this MegaCore function provides a number of configuration registers that control the format of video leaving the system (blanking period size, sync length, and interlaced or progressive mode) and a status interrupt that can be used to determine when the video format changes.

Video Formats

The Clocked Video Output MegaCore function creates the following clocked video formats:

- Video with sync information embedded in the data (in BT656 format)
- Video with separate sync (h sync, v sync) signals

The Clocked Video Output MegaCore function creates a video frame consisting of horizontal and vertical blanking (containing syncs) and areas of active picture (taken from the Avalon-ST Video input).

The format of the video frame is shown in Figure 5–14 on page 5–36 for progressive and Figure 5–15 on page 5–36 for interlaced.

Figure 5–14. Progressive Frame Format

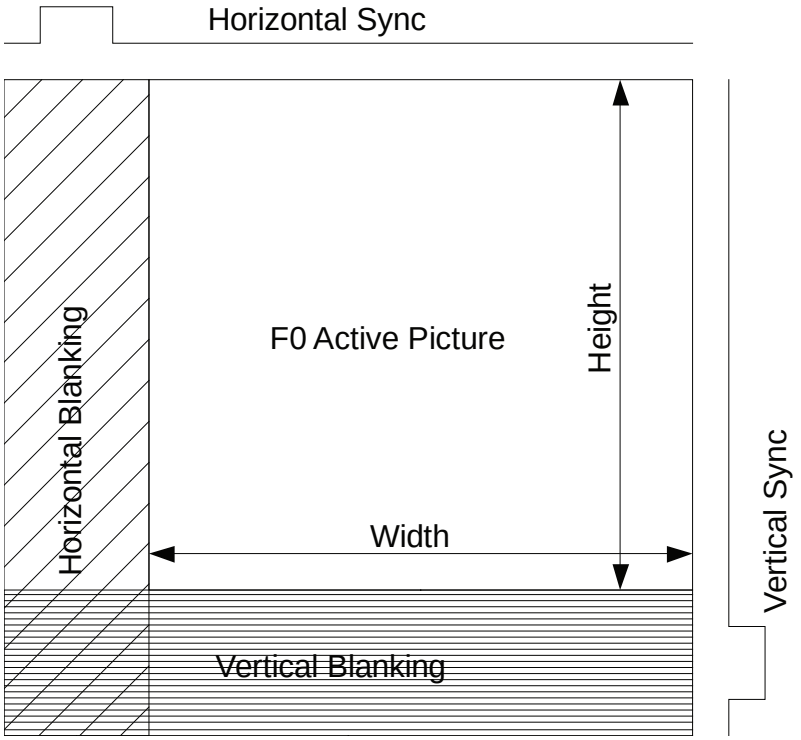
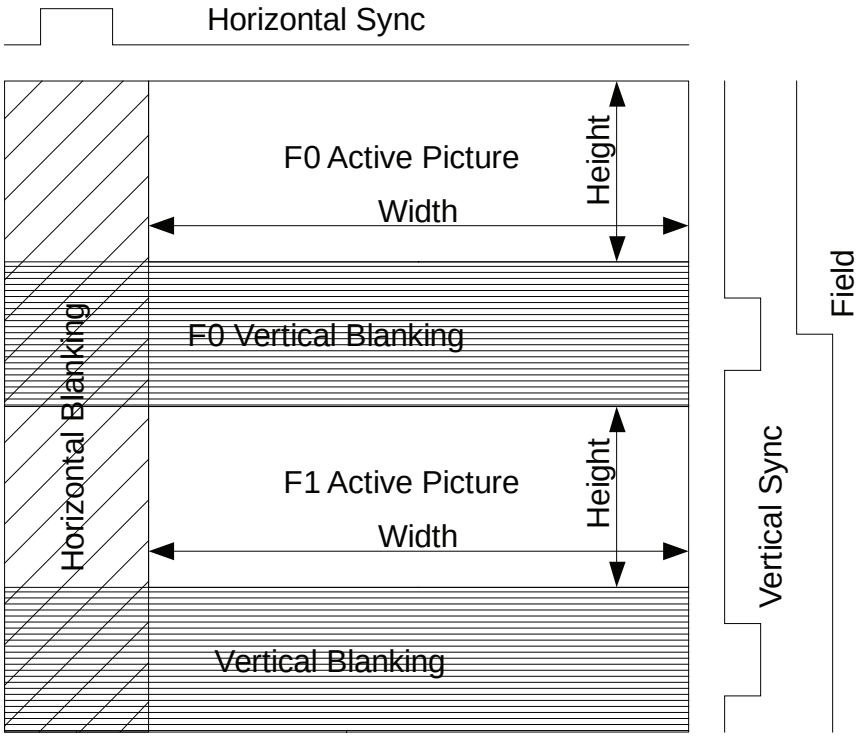


Figure 5–15. Interlaced Frame Format



Embedded Sync Format

For the embedded sync format, the MegaCore function inserts the horizontal and vertical syncs and field into the data stream during the horizontal blanking period (Table 5–12 on page 5–31).

A sample is output for each clock cycle on the `vid_data` bus. The `vid_datavalid` signal is used to indicate when the `vid_data` video output is in an active picture period of the frame.

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

Separate Sync Format

For the separate sync format, the MegaCore function outputs horizontal and vertical syncs and field information via their own signals.

A sample is output for each clock cycle on the `vid_data` bus. The `vid_datavalid` signal is used to indicate when the `vid_data` video output is in an active picture period of the frame.

Table 5–16 describes five extra signals for separate sync formats.

Table 5–16. Clocked Video Output Signals for Separate Sync Format Video

Signal Name	Description
`vid_h_sync`	1 during the horizontal sync period.
`vid_v_sync`	1 during the vertical sync period.
`vid_f`	When interlaced data is output, this is a 1 when F1 is being output and a 0 when F0 is being output. During progressive data it is always 0.
`vid_h`	1 during the horizontal blanking period.
`vid_v`	1 during the vertical blanking period.

Control Port

If you turn on **Use control port** in the MegaWizard interface for the Clocked Video Output, it can be controlled using the Avalon-MM slave control port. Initially, the MegaCore function is disabled and does not output any video. However, it still accepts data on the Avalon-ST Video interface for as long as it has space in its input FIFO.

The sequence for starting the output of the MegaCore function is as follows:

1. Write a 1 to `Control` register bit 0.
2. Read `Status` register bit 0. When this is a 1, the function outputs video.

The sequence for stopping the output of the MegaCore function is as follows:

1. Write a 0 to `Control` register bit 0.
2. Read `Status` register bit 0. When this is a 0, the function has stopped video output. This occurs at the end of the next frame or field boundary.

The starting and stopping of the MegaCore function is synchronized to a frame or field boundary.

Video Modes

The video frame is described using the mode registers that are accessed via the Avalon-MM control port. If you turn off **Use control port** in the MegaWizard interface for the Clocked Video Output, then the output video format always has the format specified in the MegaWizard interface.

The MegaCore function can be configured to support between 1 to 14 different modes and each mode has a bank of registers that describe the output frame. When the MegaCore function receives a new control packet on the Avalon-ST Video input, it searches the mode registers for a mode that is valid and has a field width and height that matches the width and height in the control packet. The register `VidModeMatch` shows the selected mode and the signal `vid_mode_match` also shows the same information. When found, it restarts the video output with those format settings. If a matching mode is not found, the video output format is unchanged and a restart does not occur.

Figure 5–16 shows how the register values map to the progressive frame format described in "Video Formats" on page 5–35.

Figure 5–16. Progressive Frame Parameters

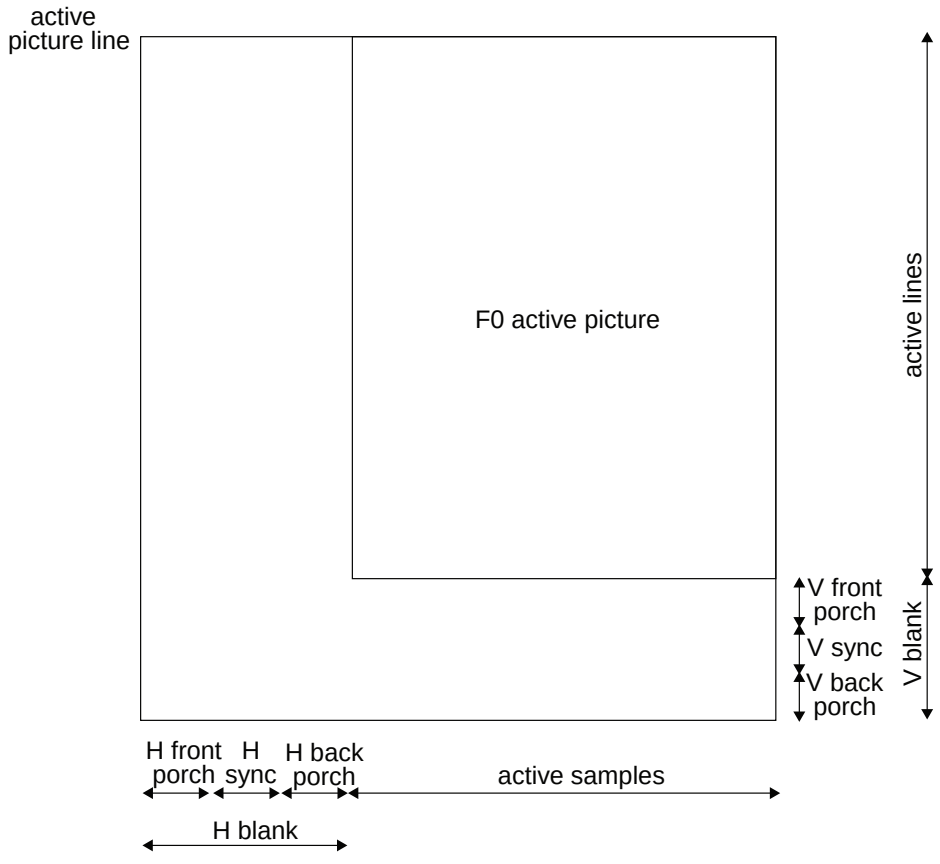


Table 5–17 on page 5–39 shows how Figure 5–16 relates to the register map.

Table 5–17. Progressive Frame Parameter Descriptions

Register Name	Parameter	Description
ModeXInterlaced	N/A	Set to 0 for progressive.
ModeXF0SampleCount	active samples	The width of the active picture region in samples/pixels.
ModeXF0LineCount	active lines	The height of the active picture region in lines.
ModeXHFrontPorch	H front porch	(Separate sync mode only.) The front porch of the horizontal sync (the low period before the sync starts).
ModeXHSyncLength	H sync	(Separate sync mode only.) The sync length of the horizontal sync (the high period of the sync).
ModeXHBlanking	H blank	The horizontal blanking period (non active picture portion of a line).
ModeXVFrontPorch	V front porch	(Separate sync mode only.) The front porch of the vertical sync (the low period before the sync starts).
ModeXVSyncLength	V sync	(Separate sync mode only.) The sync length of the vertical sync (the high period of the sync).
ModeXVBlanking	V blank	The vertical blanking period (non active picture portion of a frame).
ModeXAPLine	active picture line	The line number that the active picture starts on. For non SDI output this can be left at 0.
ModeXValid	N/A	Set to enable the mode after the configuration is complete.

Figure 5–17 on page 5–40 shows how the register values map to the interlaced frame format described in "Video Formats" on page 5–35.

Table 5–18 shows how Figure 5–17 relates to the register map.

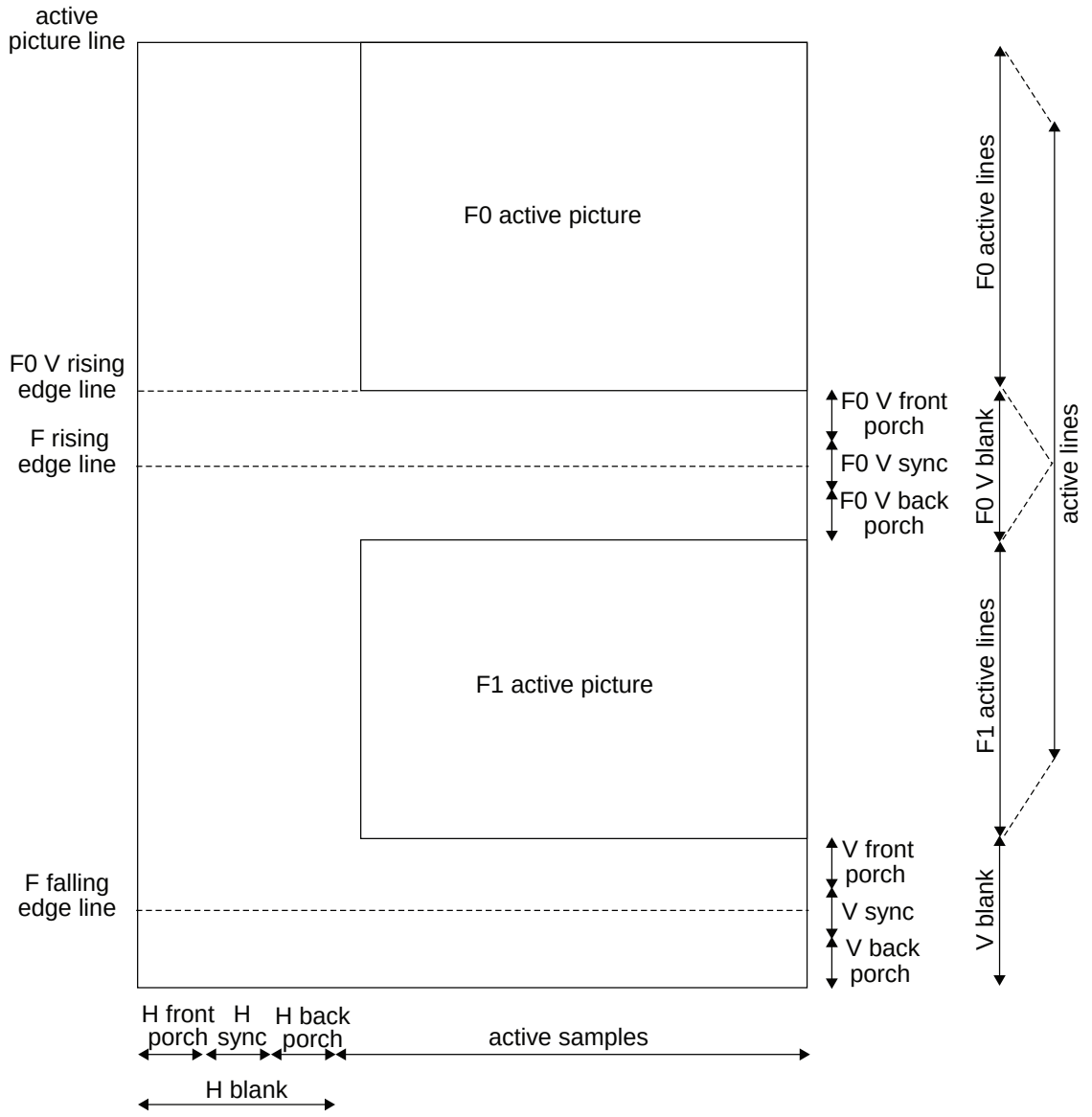
Table 5–18. Interlaced Frame Parameter Descriptions (Part 1 of 2)

Register Name	Parameter	Description
ModeXInterlaced	N/A	Set to 1for interlaced.
ModeXF0SampleCount	active samples	The width of the active picture region in samples/pixels.
ModeXF0LineCount	F0 active lines	The height of the active picture region for field F0 in lines.
ModeXF1SampleCount	active samples	The width of the active picture region in samples/pixels.
ModeXF1LineCount	F1 active lines	The height of the active picture region for field F1 in lines.
ModeXHFrontPorch	H front porch	(Separate sync mode only.) The front porch of the horizontal sync (the low period before the sync starts).
ModeXHSyncLength	H sync	(Separate sync mode only.) The sync length of the horizontal sync (the high period of the sync).
ModeXHBlanking	H blank	The horizontal blanking period (non active picture portion of a line).
ModeXVFrontPorch	V front porch	(Separate sync mode only.) The front porch of the vertical sync (the low period before the sync starts) for field F1.
ModeXVSyncLength	V sync	(Separate sync mode only.) The sync length of the vertical sync (the high period of the sync) for field F1.
ModeXVBlanking	V blank	The vertical blanking period (non active picture portion of a frame) for field F1.
ModeXVFrontPorch	F0 V front porch	(Separate sync mode only.) The front porch of the vertical sync (the low period before the sync starts) for field F0.
ModeXVSyncLength	F0 V sync	(Separate sync mode only.) The sync length of the vertical sync (the high period of the sync) for field F0.

Table 5–18. Interlaced Frame Parameter Descriptions (Part 2 of 2)

Register Name	Parameter	Description
ModeXVBlanking	F0 V blank	The vertical blanking period (non active picture portion of a frame) for field F0.
ModeXAPLine	active picture line	The line number that the active picture starts on. For non SDI output this can be left at 0.
ModeXF0VRising	F0 V rising edge line	The line number that the vertical blanking period for field F0 begins on.
ModeXFRising	F rising edge line	The line number that field F1 begins on.
ModeXFFalling	F falling edge line	The line number that field F0 begins on.
ModeXValid	N/A	Set to enable the mode after the configuration is complete.

Figure 5–17. Interlaced Frame Parameters



The mode registers can only be written to if a mode is marked as invalid. For example, the following steps reconfigure mode 1:

1. Write a 0 to the `Mode1Valid` register.
2. Write to the mode 1 configuration registers.
3. Write a 1 to the `Mode1Valid` register. The mode is now valid and can be selected.

A currently-selected mode can be configured in this way without affecting the video output of the MegaCore function.

When searching for a matching mode and there are multiple modes that match the resolution, the lowest mode is selected. For example, Mode 1 is selected over Mode 2 if they both match. To allow Mode 2 to be selected, invalidate Mode 1 by writing a 0 to its mode valid register. Invalidating a mode does not clear its configuration.

Interrupt

The Clocked Video Output MegaCore function outputs a single interrupt line. The status update interrupt triggers when the `VidModeMatch` register is updated due to a new video mode being selected.

The interrupt can be independently enabled using bit 1 of the `Control` register. The interrupt status can be read using bit 1 of the `Status` register and a write of 1 to this bit clears the interrupt.

Underflow

Moving between flow controlled Avalon-ST Video and clocked video can cause problems if the flow controlled video does not provide data at a rate fast enough to satisfy the demands of the outgoing clocked video.

The Clocked Video Output MegaCore function contains a FIFO that, when set to a large enough value, can accommodate any "burstiness" in the flow data, as long as the output rate of the downstream Avalon-ST Video components is equal to or higher than that of the outgoing clocked video.

If this is not the case, the FIFO underflows. If underflow occurs, the MegaCore function continues to output video and re-syncs the `startofpacket`, for the next image packet, from the Avalon-ST Video interface with the start of the next frame.

The underflow can be detected by looking at bit 2 of the `Status` register. This bit is sticky and if an underflow occurs, stays at 1 until the bit is cleared by writing a 1 to it. In addition to the underflow bit, the current level of the FIFO can be read from the `UsedW` register.

Timing Constraints

To constrain the Clocked Video Output MegaCore function correctly, add the following file to your Quartus II project:

<*install_dir*>\ip\clocked_video_output\lib\alt_vip_cvo.sdc.

Color Plane Sequencer

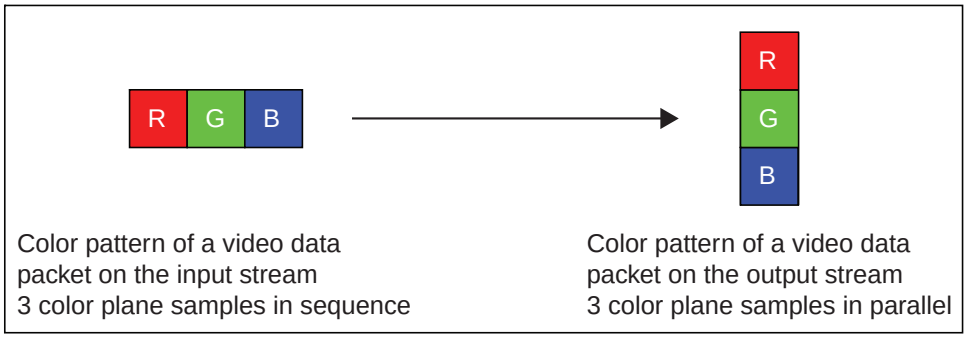
The purpose of the Color Plane Sequencer MegaCore function is to rearrange the color pattern used to transmit Avalon-ST Video data packets over an Avalon-ST connection (stream). A color pattern is a matrix that defines a repeating pattern of color samples.

For full details of the Avalon-ST Video protocol, refer to "Avalon-ST Video Protocol" on page 4–1.

The color pattern of a video data packet can be rearranged in any valid combination of channels in sequence and parallel. The Color Plane Sequencer can also drop color planes. Avalon-ST Video packets of types other than video data packets are forwarded unchanged.

Figure 5–18 shows an example that rearranges the color pattern of a video data packet which transmits color planes in sequence, to a color pattern that transmits color planes in parallel.

Figure 5–18. Example of Rearranging Color Patterns



Combining Color Patterns

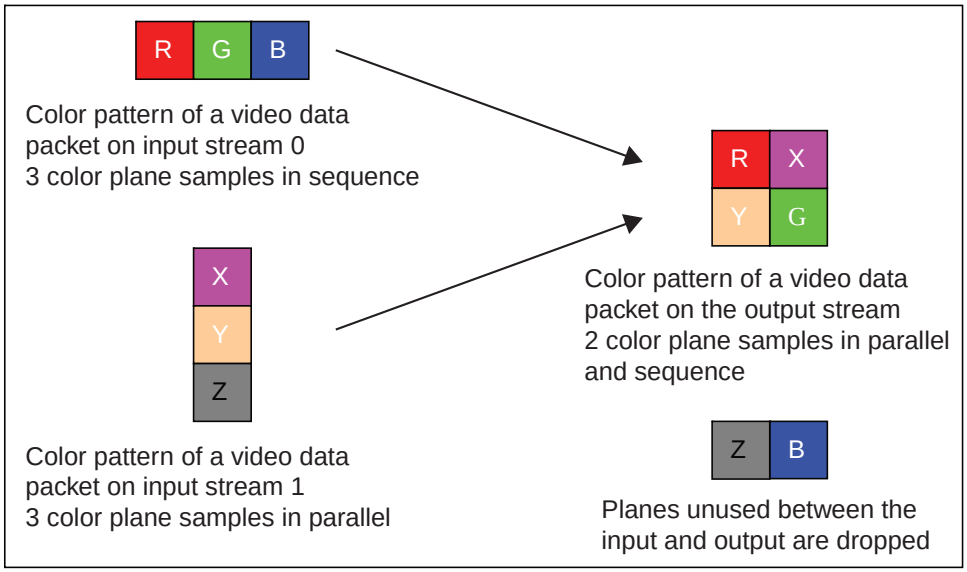
The Color Plane Sequencer also allows the combination of two Avalon-ST Video streams into a single stream. In this mode of operation, two input color patterns (one for each input stream) are combined and arranged to the output stream color pattern in a user defined way, so long as it contains a valid combination of channels in sequence and parallel.

In addition to this combination and arrangement, color planes can also be dropped. Avalon-ST Video packets other than video data packets can be forwarded to the single output stream with the following options:

- Packets from input stream 0 (port `din0`) and input stream 1 (port `din1`) forwarded, input stream 0 packets being transmitted last. (The last control packet received is the one an Avalon-ST Video compliant MegaCore function uses.)
- Packets from input stream 0 forwarded, packets from input stream 1 dropped.
- Packets from input stream 1 forwarded, packets from input stream 0 dropped.

Figure 5–19 shows an example of combining and rearranging two color patterns.

Figure 5–19. Example of Combining Color Patterns



Splitting/Duplicating

The Color Plane Sequencer also allows the splitting of a single Avalon-ST Video input stream into two Avalon-ST Video output streams. In this mode of operation, the color patterns of video data packets on the output streams can be arranged in a user defined way using any of the color planes of the input color pattern.

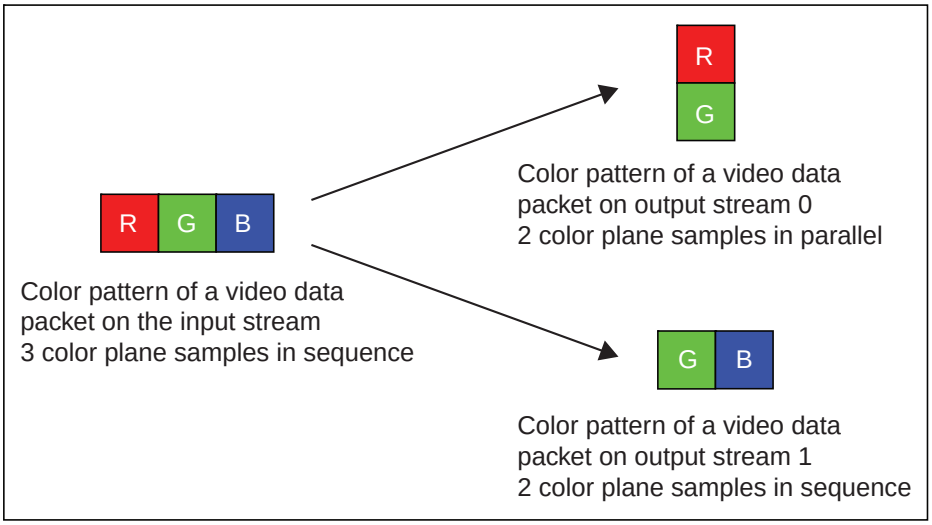
The color planes of the input color pattern are available for use on either, both, or neither of the outputs. This allows for splitting of video data packets, duplication of video data packets, or a mix of splitting and duplication.

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.

Avalon-ST Video packets other than video data packets are duplicated to both outputs.

Figure 5–20 on page 5–44 shows an example of partially splitting and duplicating an input color pattern.

Figure 5–20. Example of Splitting and Duplicating Color Patterns



Subsampled Data

In addition to fully sampled color patterns, the Color Plane Sequencer supports 4:2:2 subsampled data. To facilitate this support, you can configure the Color Plane Sequencer with two color patterns in sequence, so that subsampled planes can be specified individually.

When splitting subsampled planes from fully-sampled planes, the Avalon-ST Video control packet for the subsampled video data packet can have its width value halved, so that the subsampled planes can be processed by other MegaCore functions as if fully sampled. This halving can be applied to control packets on port `dout0` and port `dout1`, or control packets on port `dout0` only.

Avalon-ST Video Stream Requirements

The only stream requirement imposed is that when two streams are being combined, the video data packets must contain the same total number of pixels, and to make a valid image, the packets must have the same dimensions.

The Color Plane Sequencer MegaCore function can process streams of pixel data of the types shown in Table 5–19.

Table 5–19. Color Plane Sequencer Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	Read from control packets at run time.
Frame Height	Read from control packets at run time.
Interlaced/Progressive	Either.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color Pattern	The color pattern is selected in the MegaWizard interface.

Test Pattern Generator

The Test Pattern Generator MegaCore function can be used to produce a video stream compliant with the Avalon-ST Video protocol that feeds a video system during its design cycle.

The Test Pattern Generator MegaCore function 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 MegaCore function or of a custom component using a genuine video input.

Generation of Avalon-ST Video Control Packets and Run-Time Control

The Test Pattern Generator MegaCore function outputs a valid Avalon-ST Video control packet before each image data packet it generates, whether it is a progressive frame or an interlaced field. When the output is interlaced, the Test Pattern Generator MegaCore function produces a sequence of pairs of field, starting with F0 if the output is F1 synchronized of with F1 if the output is F0 synchronized.

When the Avalon Slave run-time controller is enabled, 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. For details of the control register map for the Test Pattern Generator, refer to Table A–35 on page A–26.

Because the Test Pattern Generator does not accept an input stream, the pseudo-code in "Avalon-MM Slave Interfaces" on page 4–14 is slightly modified:

```
go = 0;
while (true)
{
    status = 0;
    while (go != 1 )
        wait();
    send_control();     //Copies control to internal register
    status = 1;
    send_control_packet();
    send_image_data_header();
    output_test_pattern ();
}
```

Output Data Types

The Test Pattern Generator MegaCore function supports a wide range of resolutions and color spaces with either a sequential or parallel data interface.

In all combinations of color space and subsampling that are allowed, the stream of pixel data is of a type consistent with the conventions adopted by the other MegaCore functions in the Video and Image Processing Suite.

The Test Pattern Generator function can output streams of pixel data of the types shown in Table 5–20 on page 5–46.

Table 5–20. Test Pattern Generator Avalon-ST Video Protocol Parameters

Parameter	Value
Frame Width	Width selected in the MegaWizard interface. Can be run-time controlled in which case, the value specified in the GUI is the maximum allowed value.
Frame Height	Height selected in the MegaWizard interface. Can be run-time controlled in which case, the value specified in the GUI is the maximum allowed value.
Interlaced/Progressive	Mode selected in the MegaWizard interface.
Bits per Color Sample	Number of bits per color sample selected in the MegaWizard interface.
Color space	As selected in the MegaWizard interface. RGB (4:4:4 subsampling only) or YCbCr.
Color Pattern	For RGB sequential data:  For RGB parallel data:  For 4:4:4 sequential data:  For 4:2:2 sequential data:  For 4:2:0 sequential data:  For 4:2:2 parallel data:  For 4:4:4 parallel data:  For 4:2:0 parallel data: 

Notes to Table 5–20:

(1) 4:2:2 and 4:2:0 subsampling are not available for the RGB color space.

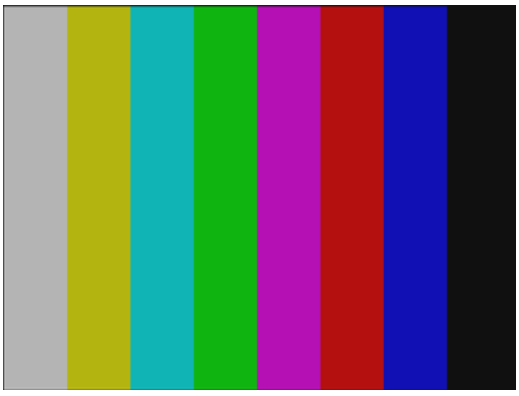
(2) Vertical subsampling and interlacing cannot be used when the height of the output is not even. The GUI does not allow such a parameterization and the behavior of the MegaCore function is undefined if the height is subsequently set to an odd value through the run-time control.

(3) Vertical subsampling and interlacing are incompatible with each other and cannot be selected simultaneously in the GUI.

Test Pattern

The Test Pattern Generator MegaCore function can generate either a uniform image using a constant color specified by the user at compile time or a set of predefined color bars. The color bar pattern (Figure 5–21) is a still image composed with a set of eight vertical color bars of 75% intensity (white, yellow, cyan, green, magenta, red, blue, black). The pattern is delimited by a thin black rectangular border.

Figure 5–21. Generated Test Pattern



The sequence runs through the eight possible on/off combinations of the three color components of the RGB color space starting with a 75% amplitude white. Green is on for the first four bars and off for the last four bars, red cycles on and off every two bars, and blue cycles on and off every bar.

The actual numerical values are given in Table 5–21 (assuming 8 bits per color samples). If the output is requested in a different number of bits per color sample these values are converted by truncation or promotion.

Table 5–21. Test Pattern Color Values

	R'G'B'	Y'CbCr
White/Grey	(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 choice of a specific resolution and subsampling for the output leads to natural constraints on the test pattern. If the format has a horizontal subsampling period of two for the Cb and Cr components when the output is in the Y'CbCr color space, the black borders at the left and right are two pixels wide. Similarly, the top and bottom borders are two pixels wide when the output is vertically subsampled.

The width and the horizontal subsampling might also have an effect on the width of each color bar. 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).

Stall Behavior

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

During control packet processing, the MegaCore functions may stall frequently and read/write less than once per clock cycle. During data processing, the MegaCore functions generally process one input/output per clock cycle. There are, however, some stalling cycles. Typically, these are for internal calculations between rows of image data and between frames/fields.

When stalled, the MegaCore function signals that it is not ready to receive or produce data. The time spent in the stalled state varies between MegaCore functions and their parameterizations. In general, it is a few cycles between rows and a few more between frames. Details of exceptions to this behavior and details of stalling due to internal buffering are given for each MegaCore function in the following sections.

When they are not stalled, all the Video and Image Processing Suite MegaCore functions process one sample on every clock cycle (rate-changing functions process one sample on the higher-rate side on every clock cycle).

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

The following sections give bounds and guidelines describing the stalling and throughput of the MegaCore functions but do not attempt to specify precise behavior down to the last clock cycle. When an `endofpacket` signal is received unexpectedly (early or late), the MegaCore function recovers from the error and prepares itself for the next valid packet (control or data). The time taken to do this is described in each of the following sections.

The exact behavior of the MegaCore functions may vary between releases or if any of the parameters are changed.

Color Space Converter

In all parameterizations, the Color Space Converter only stalls between frames and not between rows. It has no internal buffering apart from the registers of its processing pipeline so there are few clock cycles of latency.

Error Recovery

The Color Space Converter MegaCore function processes video packets until an endofpacket signal is received; the control packets are not used. For this MegaCore function, there is no such condition as an early or late endofpacket, any mismatch of the endofpacket signal and the frame size is propagated unchanged to the next MegaCore function.

Chroma Resampler

All modes of the Chroma Resampler stall for a few cycles between frames and between lines.

Latency from input to output varies depending on the operation mode of the Chroma Resampler MegaCore function. 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. These modes have a latency corresponding to two lines worth of 4:2:0 data.

Because this is a rate-changing function, the quantities of data input and output are not equal. The Chroma Resampler MegaCore function always outputs the same number of lines that it inputs. However the number of samples in each line varies according to the subsampling pattern used.

When not stalled, the Chroma Resampler always processes one sample from the more fully sampled side on each clock cycle. For example, the subsampled side 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.

Error Recovery

On receiving an early endofpacket signal, the Chroma Resampler stalls its input but continues writing data until it has sent an entire frame. If it does not receive an endofpacket signal at the end of a frame, the Chroma Resampler discards data until the end of packet is found.

Gamma Corrector

In all parameterizations, the Gamma Corrector stalls only between frames and not between rows. It has no internal buffering aside from the registers of its processing pipeline so there are few clock cycles of latency.

Error Recovery

The Gamma Corrector MegaCore function processes video packets until an endofpacket signal is received; the control packets are not used. For this MegaCore function there is no such condition as an early or late endofpacket. Any mismatch of the endofpacket signal and the frame size is propagated unchanged to the next MegaCore function.

2D FIR Filter

There is a delay of a little more than N–1 lines between data input and output in the case of a $N \times N$ 2D FIR Filter. This is due to line buffering internal to the MegaCore function.

Error Recovery

The 2D FIR Filter MegaCore function resolution is not configurable at runtime. This MegaCore function does not read the control packets passed through it.

An error condition occurs if an `endofpacket` signal is received too early or too late for the compile time configured frame size. In either case, the 2D FIR Filter always creates output video packets of the configured size. 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, the video frame is padded with an undefined combination of the last input pixels.

2D Median Filter

There is a delay of a little more than N–1 lines between data input and output in the case of a $N \times N$ 2D Median Filter. This is due to line buffering internal to the MegaCore function.

Error Recovery

The 2D Median Filter MegaCore function resolution is not configurable at run time. This MegaCore function does not read the control packets passed through it.

An error condition occurs if an `endofpacket` signal is received too early or too late for the compile-time-configured frame size. In either case, the 2D FIR Filter always creates output video packets of the configured size.

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, the video frame is padded with an undefined combination of the last input pixels.

Alpha Blending Mixer

For each non-stalled cycle, the Alpha Blending Mixer reads from the background input port, and also from the input port associated with each layer which covers the background pixel just read.

When alpha blending is enabled, data 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 line buffering in the Alpha Blending Mixer MegaCore function, so the delay from input to output is just a few clock cycles caused by pipelining.

Scaler

In the Scaler MegaCore function, 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 Scaler depends on the scaling algorithm and whether any run time control is enabled. The scaling algorithm impacts stalling as follows:

- In nearest-neighbor mode, the delay from input to output is just a few clock cycles.
- In 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. Exactly how many writes are possible during this time depends on the scaling ratio.
- In bicubic mode, three lines of input are read into line buffers before any output is ready. As with linear interpolation, there is a scaling ratio dependent time at the end of a frame where no reads are needed as the buffers are drained.
- In polyphase mode with N_v vertical taps, $N_v - 1$ lines of input are read into line buffers before any output is ready. As with bilinear mode, there is a scaling ratio dependent time at the end of a frame where no reads are needed as the buffers are drained.

Enabling run-time control of coefficients and/or resolutions affects stalling between frames:

- With no run-time control, there is only a few cycles of delay before the behavior described in the previous list begins.
- Enabling run-time control of resolutions in nearest-neighbor mode adds about 20 clock cycles of delay between frames. In other modes, it adds a maximum of 60 cycles delay.
- Enabling run-time control of coefficients adds a constant delay of about 20 cycles plus the total number of coefficients to be read. For example, 16 taps and 32 phases in each direction would add a delay of 20 + 2(16 × 32) = 1024 cycles.

Error Recovery

On receiving an early `endofpacket` signal, the Scaler stalls its input but continues writing data until it has sent an entire frame. If it does not receive an `endofpacket` signal at the end of a frame, the Scaler discards data until the end-of-packet is found.

Clipper

The Clipper MegaCore function stalls for a few cycles between lines and between frames. Its internal latency is less than 10 cycles. During the processing of a line, it reads continuously but the Clipper only writes when inside the active picture area as defined by the clipping window.

Error Recovery

On receiving an early `endofpacket` signal, the Clipper stalls its input but continues writing data until it has sent an entire frame. If it does not receive an `endofpacket` signal at the end of a frame, the Clipper discards data until the end-of-packet is found.

Deinterlacer

While the bob algorithm (with no buffering) is producing an output frame it alternates between simultaneously receiving a row on the input port and producing a row of data on the output port, and just producing a row of data on the output port without reading any data from the input port.

The delay from input to output is just a few clock cycles. While a field is being discarded, input is read at the maximum rate and no output is generated.

When the weave algorithm is selected, the MegaCore function stalls for longer than the usual periods between each output row of the image. Stalls of up to 45 clock cycles are possible due to the time taken for internal processing in between lines.

When the motion-adaptive algorithm is selected, stalls up to 90 clock cycles are possible.

When double or triple-buffering is selected, data input and output are decoupled through the use of external memory. The Megacore function writes non-image data packets into memory by pre-declaring transfers of fixed size and memory transactions cannot be interrupted immediately when an `endofpacket` signal is received.

For each non-image data packet received, the number of words written into memory always corresponds to the maximum packet size defined in the MegaWizard interface. Consequently, the Deinterlacer Megacore function does not handle control packets efficiently when large user-defined packets are used. This does not apply when reading non-image packets back from the external memory because the size of each incoming packet is registered after it has been determined.

When buffering is used with bob deinterlacing and fields are being discarded they are discarded at the input rather than being buffered through external RAM and then discarded. This reduces the external RAM bandwidth requirement of the Deinterlacer in these modes.

Error Recovery

An error condition occurs if an `endofpacket` signal is received too early or too late relative to the field dimensions contained in the last control packet processed. In all its configurations, the Deinterlacer discards extra data if the `endofpacket` signal is received too late.

If an early `endofpacket` signal is received when the Deinterlacer is configured for no buffering, the MegaCore function interrupts its processing within one or two lines sending undefined pixels, before propagating the `endofpacket` signal.

If an early `endofpacket` signal is received when the Deinterlacer is configured to buffer data in external memory, the input side of the MegaCore function stops processing input pixels. It is then ready to process the next frame after writing undefined pixels for the remainder of the current line into external RAM. The output side of the Deinterlacer assumes that incomplete fields have been fully received and pads the incomplete fields to build a frame, using the undefined content of the memory.

Frame Buffer

The Frame Buffer Megacore function writes data into memory by pre-declaring transfers of fixed size. Memory transactions cannot be interrupted immediately and extra data may be written into memory after an `endofpacket` signal has been received.

For each non-image data packet received, the number of words written into memory always corresponds to the maximum packet size defined in the MegaWizard interface. The Megacore function does not handle control packets efficiently when large user-defined packets are used.

When processing an image data packet, the number of words written into memory for each transaction is matched with the maximum size of a line as defined in the MegaWizard interface. This might introduce extra latency between frames and a small memory bandwidth overhead when the input resolution does not match with the maximum resolution defined in the MegaWizard interface.

The Frame Buffer registers the size of each incoming Avalon-ST Video packet and the stalls described above do not apply when reading non-image and image data packets from the external memory.

The Frame Buffer stalls for a few cycles between memory transactions and stalls in case of bus contention.

Error Recovery

The Frame Buffer Megacore function does not process the control packets passed through it and there is consequently no error condition such as early or late `endofpacket` signal. However, the Frame Buffer does not write outside the memory allocated for each non-image and image data packets and input is discarded at the maximum rate in case of overflow.

Color Plane Sequencer

The Color Plane Sequencer MegaCore function stalls for approximately 10 cycles after processing each line of a video frame. Between frames the MegaCore function stalls for approximately 30 cycles.

Error Recovery

The Color Plane Sequencer MegaCore function processes video packets per line until an `endofpacket` signal is received on `din0`. (The line width is taken from the control packets on `din0`.)

When an `endofpacket` signal is received on either `din0` or `din1` the Color Plane Sequencer ceases output. For the number of cycles left to finish the line, the MegaCore function continues to drain the inputs that have not indicated end-of-packet.

The MegaCore function drains `din0` until it receives an `endofpacket` signal on this port (unless it has already indicated end-of-packet), and stalls for up to one line after this `endofpacket` signal. The MegaCore function then signals end-of-packet on its outputs and continue to drain its inputs that have not indicated end-of-packet.

Test Pattern Generator

All modes of the Test Pattern Generator stall for a few cycles after a field, after a control packet, and between lines. When producing a line of image data, the Test Pattern Generator outputs one sample on every clock cycle, but it can be stalled without consequences if other functions down the data path are not ready and exert backpressure.

Latency

Table 5–22 on page 5–54 shows the approximate latency from the video data input to the video data output for typical usage modes of each MegaCore function. You can use this table 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 video frames
- the number of video fields
- the number of lines when less than a field of latency
- a small number of cycles *O* (cycles)

Table 5–22. Latency Summary

MegaCore Function	Mode	Latency *(Note 1)*
Color Space Converter	All modes	*O* (cycles)
Chroma Resampler	Input format: 4:2:2 Output format: 4:4:4	*O* (cycles)
Gamma Corrector	All modes	*O* (cycles)
2D FIR Filter	Filter size: *N* × *N*	(*N*–1) lines +*O* cycles
2D Median Filter	Filter size: *N* × *N*	(*N*–1) lines +*O* cycles
Alpha Blending Mixer	All modes	*O* (cycles)
Scaler	Scaling algorithm: Polyphase Number of vertical taps: *N*	(*N*–1) lines +*O* cycles
Clipper	All modes	*O* (cycles)
Deinterlacer	Method: Bob Frame buffering: None	*O* (cycles)
	Method: Motion-adaptive or Weave Frame buffering: Double or triple buffering with rate conversion Output frame rate: As input frame rate	1 field +*O* lines
	Method: Motion-adaptive or Weave Frame buffering: Double or triple buffering with rate conversion Output frame rate: As input field rate	1 frame +*O* lines
	Method: All Frame buffering: Double or triple buffering with rate conversion Passthrough mode (propagate progressive frames unchanged): On.	1 frame +*O* lines
Frame Buffer	All modes	1 frame +*O* lines
Color Plane Sequencer	All modes	*O* (cycles)
Clocked Video Input *(Note 2)*	Sync signals: Embedded in video Video in and out use the same clock: On	8 cycles
	Sync signals: On separate wires Video in and out use the same clock: On	5 cycles
Clocked Video Output *(Note 2)*	All modes with Video in and out use the same clock: On	3 cycles *(Note 3)*
Test Pattern Generator	Not Applicable because the Test Pattern Generator is an Avalon-ST Video source only.	N/A

Notes to Table 5–22:

(1) It is assumed that the MegaCore function is not being stalled by other functions on the data path (the output ready signal is high).

(2) Add 1 cycle if **Allow color planes in sequence input** is turned on.

(3) Minimum latency case when video input and output rates are synchronized.

The latency associated with the initial buffering phase, when a MegaCore function first receives video data, is not included. For example, the Deinterlacer MegaCore function in motion-adaptive mode initially buffers four fields of video in external memory without outputting data. After the initial buffering phase, the latency from field input to frame output (assuming the output frame rate is the same as the input field rate) is one field + *O* (lines).

Compile Time Parameters

Table A–1 to Table A–15 describe the Video and Image Processing Suite MegaCore function parameters. You can set these parameters in the MegaWizard interface as described in "Parameter Settings" on page 3–1.

The default parameter values are shown using bold text in the tables.

Color Space Converter

Table A–1 and Table A–2 show the Color Space Converter MegaCore function parameters.

Table A–1. Color Space Converter Parameter Settings, General Tab (Part 1 of 2)

Parameter	Value	Description
Color Plane Configuration	**Three color planes in sequence**, or Three color planes in parallel	There must always be three color planes for this function but you can choose whether the three color planes are transmitted in sequence or in parallel.
Input Data Type: Bits per pixel per color plane	4–20, Default = **8**	Choose the number of input bits per pixel (per color plane).
Input Data Type: Data type	**Unsigned**, Signed	Specify whether the input is unsigned or signed 2's complement.
Input Data Type: Guard bands	On or **Off**	Turn on to enable 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**	Choose the number of output bits per pixel (per color plane).
Output Data Type: Data type	**Unsigned**, Signed	Specify whether the output is unsigned or 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.
Move binary point right	–16 to +16, Default = **0**	Specify the number of places to move the binary point.

Table A–1. Color Space Converter Parameter Settings, General Tab (Part 2 of 2)

Parameter	Value	Description
Remove fraction bits by	**Round values - Half up**, Round values - Half even, Truncate values to integer	Choose 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	Choose the method of signed to unsigned conversion for the results.

Table A–2. Color Space Converter Parameter Settings, Operands Tab

Parameter	Value	Description
Color model conversion	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', Custom	You can select a predefined set of coefficients and summands which are used for color model conversion at compile time. Alternatively, you can create your own custom set by modifying the *din_0*, *din_1*, and *din_2* coefficients for *dout_0*, *dout_1*, and *dout_2* separately. The values are assigned in the order indicated by the conversion name. For example, if you select **Computer B'G'R' to CbCrY': SDTV**, then *din_0* = B', *din_1* = G', *din_2* = R', *dout_0* = Cb, *dout_1* = Cr, and *dout_2* = Y'.
Runtime controlled	On or **Off**	Turn on to enable run-time control of the conversion values.
Coefficients and Summands 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 purple cell underneath. The value in the white cell is the desired value, and is editable. The value in the purple cell is the actual value, determined by the fixed-point type specified. The purple cells are not editable. You can create a custom coefficient and summand set by specifying one fixed-point value for each entry.
Coefficients: Signed	**On** or Off	Turn on to set the fixed point type used to store the constant coefficients as having a sign bit.
Coefficients: Integer bits	8–31, Default = **0**	Specifies the number of integer bits for the fixed point type used to store the constant coefficients.
Summands: Signed	On or **Off**	Turn on to set the fixed point type used to store the constant summands as having a sign bit.
Summands: Integer bits	8–31, Default = **8**	Specifies the number of integer bits for the fixed point type used to store the constant summands.
Coefficient and summand fraction bits	8–31, Default = **8**	Specifies the number of fraction bits for the fixed point type used to store the coefficients and summands.

Chroma Resampler

Table A–3 shows the Chroma Resampler MegaCore function parameters.

Table A–3. Chroma Resampler Parameter Settings

Parameter	Value	Description
Maximum width	32–2,600, Default = **256**	Choose the maximum image width in pixels.
Maximum height	32–2,600, Default = **256**	Choose the maximum image height in pixels.
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Color plane configuration	Sequence, Parallel	There must always be three color planes for this function but you can choose whether the three color planes are transmitted in sequence or in parallel.
Input Format	4:4:4, **4:2:2**, 4:2:0	Choose the format/sampling rate format for the input frames. Note that the input and output formats must be different.
Output Format	**4:4:4**, 4:2:2, 4:2:0	Choose the format/sampling rate format for the output frames. Note that the input and output formats must be different.
Horizontal Filtering Algorithm	**Filtered**, Nearest Neighbor	Choose the algorithm to use in the horizontal direction when re-sampling data to or from 4:4:4.
Luma adaptive	**On** or Off	Turn on to enable luma-adaptive mode. This mode looks at the luma channel during interpolation and uses this to detect edges.

Gamma Corrector

Table A–4 shows the Gamma Corrector MegaCore function parameters.

Table A–4. Gamma Corrector Parameter Settings

Parameter	Value	Description
Bits per pixel per color plane	4–16, Default = **8**	Choose the number of bits per pixel (per color plane).
Number of color planes	1– **3**	The number of color planes that are sent in sequence or parallel over one data connection.
Color plane transmission format	**Color planes in sequence**, Color planes in parallel	Specifies whether the specified number of color planes are transmitted in sequence or in parallel. For example, a value of 3 planes in sequence for R'G'B' R'G'B' R'G'B'.

2D FIR Filter

Table A–5 and Table A–6 on page A–4 show the 2D FIR Filter MegaCore function parameters.

Table A–5. 2D FIR Filter Parameter Settings, General Tab (Part 1 of 2)

Parameter	Value	Description
Maximum image width	32–2,600, Default = **640**	Choose the maximum image width in pixels.
Number of color planes in sequence	1–**3**	The number of color planes that are sent in sequence over one data connection. For example, a value of 3 for R'G'B' R'G'B' R'G'B'.

Table A–5. 2D FIR Filter Parameter Settings, General Tab (Part 2 of 2)

Parameter	Value	Description
Input Data Type: Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Input Data Type: Data type:	**Unsigned**, Signed	Choose whether input is unsigned or signed 2's complement.
Input Data Type: Guard bands	On or **Off**	Turn on to enable a defined input range.
Input Data Type: Max	1,048,575 to -524,288, Default = **255**	Set input range maximum value. *(Note 1)*
Input Data Type: Min	1,048,575 to -524,288, Default = **0**	Set input range minimum value. *(Note 1)*
Output Data Type: Data type	**Unsigned**, Signed	Choose whether output is unsigned or signed 2's complement.
Output Data Type: Guard bands	On or **Off**	Turn on to enable a defined output range.
Output Data Type: Max	1,048,575 to -524,288, Default = **255**	Set output range maximum value. *(Note 2)*
Output Data Type: Min	1048575 to -524288, Default = **0**	Set output range minimum value. *(Note 2)*
Move binary point right	–16 to +16, Default = **0**	Specify the number of places to move the binary point. This can be useful if you require a wider range output on an existing coefficient set.
Remove fraction bits by	**Round values - Half up**, Round values - Half even, Truncate values to integer	Choose the method for discarding fractional bits resulting from the FIR calculation.
Convert from signed to unsigned by	**Saturating to minimum value at stage 4**, Replacing negative with absolute value	Choose the method for signed to unsigned conversion of the FIR results.

Notes to Table A–5

(1) The maximum and minimum guard bands values specify a range in which the input should always fall. The 2D FIR filter behaves unexpectedly for values outside this range.

(2) The output is constrained to fall in the specified range of maximum and minimum guard bands values.

Table A–6. 2D FIR Filter Parameter Settings, Coefficients Tab (Part 1 of 2)

Parameter	Value	Description
Filter size	**3x3**, 5x5, 7x7	Choose the size in pixels of the convolution kernel used in the filtering.
Runtime controlled	**On** or Off	Turn on to enable run-time control of the coefficient values.
Coefficient set	**Simple Smoothing**, Simple Sharpening, Custom	You can choose a predefined set of simple smoothing or simple sharpening coefficients which are used for color model convolution at compile time. Alternatively, you can create your own custom set of coefficients by modifying the coefficients in the matrix.

Table A–6. 2D FIR Filter Parameter Settings, Coefficients Tab (Part 2 of 2)

Parameter	Value	Description
Enable symmetric mode	**On** or Off	When on, only symmetric coefficients are allowed. This option enables an optimization in the hardware which reduces the number of multiplications required. In this mode a limited number of matrix cells are editable and many of the values are automatically inferred. Symmetric mode is enabled for the predefined coefficient sets but can be disabled when setting custom coefficients. If you turn off this option while one of the predefined coefficient sets is selected, its values are used as the defaults for a new custom set.
Coefficients	9, 25, or 49 fixed-point values	Each coefficient is represented by a white box with a purple box underneath. The value in the white box is the desired coefficient value, and is editable. The value in the purple box is the actual coefficient value as determined by the coefficient fixed point type specified. The purple boxes are not editable. You can create a custom set of coefficients by specifying one fixed-point value for each entry in the convolution kernel. The matrix size depends on the selected filter size.
Coefficient Precision: Signed	On or **Off**	Turn on if you want the fixed-point type that stores the coefficients to have a sign bit.
Coefficient Precision: Integer bits	0–35, Default = **0**	Specifies the number of integer bits for the fixed-point type used to store the coefficients.
Coefficient Precision: Fraction bits	0–35, Default = **6**	Specifies the number of fractional bits for the fixed point type used to store the coefficients.

2D Median Filter

Table A–7 shows the 2D Median Filter MegaCore function parameters.

Table A–7. 2D Median Filter Parameter Settings

Parameter	Value	Description
Image width	32–2,600, Default = **640**	Choose the required image width in pixels.
Image height	32–2,600, Default = **480**	Choose the required image height in pixels.
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Number of color planes in sequence	1–**3**	The number of color planes that are sent in sequence over one data connection. For example, a value of 3 for R'G'B' R'G'B' R'G'B'.
Filter size	**3x3**, 5x5, 7x7	Choose the size of kernel in pixels to take the median from.

Alpha Blending Mixer

Table A–8 shows the Alpha Blending Mixer MegaCore function parameters.

Table A–8. Alpha Blending Mixer Parameter Settings (Part 1 of 2)

Parameter	Value	Description
Maximum layer width	32–2,600, Default = **1,024**	Choose the maximum image width for the layer background in pixels. No layer width can be greater than the background layer width. The maximum image width is the default width for all layers at start-up.
Maximum layer height	32–2,600, Default = **768**	Choose the maximum image height for the layer background in pixels. No layer height can be greater than the background layer height. The maximum image height is the default height for all layers at start-up.

Table A–8. Alpha Blending Mixer Parameter Settings (Part 2 of 2)

Parameter	Value	Description
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Number of color planes in sequence	1–**3**	Choose the number of color planes that are sent in sequence over one data connection. For example, a value of 3 for R'G'B' R'G'B' R'G'B'.
Number of color planes in parallel	**1**–3	Choose the number of color planes in parallel.
Number of layers being mixed	**2**–12	Choose the number of image layers to overlay. Higher number layers are mixed on top of lower layer numbers. The background layer is always layer 0.
Alpha blending	On or **Off**	When on, alpha data sink ports are generated for each layer (including the background layer). This requires a stream of alpha values; one value for each pixel. When off, no alpha data sink ports are generated, and the image layers are fully opaque.
Alpha bits per pixel	2, 4, **8**	Choose the number of bits used to represent the alpha coefficient.

Scaler

Table A–9, Table A–10, and Table A–11 on page A–7 show the Scaler MegaCore function parameters.

Table A–9. Scaler Parameter Settings, Resolution Tab

Parameter	Value	Description
Run-time control of image size	On or **Off**	Turn on to enable run-time control of the image size. When on, the input and output size parameters control the maximum values. When off, the Scaler does not respond to changes of resolution in control packets.
Input image width	32–2,600, Default = **1,024**	Choose the required input width in pixels.
Input image height	32–2,600, Default = **768**	Choose the required input height in pixels.
Output image width	32–2,600, Default = **640**	Choose the required output width in pixels.
Output image height	32–2,600, Default = **480**	Choose the required output height in pixels.
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Number of color planes	1–3, Default = **3**	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 transmission format	**Sequence**, Parallel	The transmission mode used for the specified number of color planes.

Table A–10. Scaler Parameter Settings, Algorithm and Precision Tab (Part 1 of 2)

Parameter	Value	Description
Scaling Algorithm	Nearest Neighbor, Bilinear, Bicubic, **Polyphase**	Choose the scaling algorithm. For more information about these options, refer to pages 5–12 to 5–14.
Number of vertical taps	3–16, Default = **4**	Specify the number of vertical taps.

Table A–10. Scaler Parameter Settings, Algorithm and Precision Tab (Part 2 of 2)

Parameter	Value	Description
Number of vertical phases	2, 4, 8, **16**, 32, 64, 128, 256	Specify the number of vertical phases.
Number of horizontal taps	3–16, Default = **4**	Specify the number of horizontal taps.
Number of horizontal phases	2, 4, 8, **16**, 32, 64, 128, 256	Specify the number of horizontal phases.
Vertical Coefficient Precision: Signed	**On** or Off	Turn on if you want the fixed-point type that stores the vertical coefficients to have a sign bit.
Vertical Coefficient Precision: Integer bits:	0–15, Default = **1**	Specifies the number of integer bits for the fixed-point type used to store the vertical coefficients.
Vertical Coefficient Precision: Fraction bits:	3–15, Default = **7**	Specifies the number of fractional bits for the fixed point type used to store the vertical coefficients.
Number of bits to preserve between vertical and horizontal filtering	3–32, Default = **9**	Specifies the number of bits to preserve between vertical and horizontal filtering.
Horizontal Coefficient Precision: Signed	**On** or Off	Turn on if you want the fixed-point type that stores the horizontal coefficients to have a sign bit.
Horizontal Coefficient Precision: Integer bits:	0–15, Default = **1**	Specifies the number of integer bits for the fixed-point type used to store the horizontal coefficients.
Horizontal Coefficient Precision: Fraction bits:	0–15, Default = **7**	Specifies the number of fractional bits for the fixed point type used to store the horizontal coefficients.

Table A–11. Scaler Parameter Settings, Coefficients Tab (Part 1 of 2)

Parameter	Value	Description
Load coefficient data at runtime	On or **Off**	Turn on to load the coefficient data at runtime.
Share horizontal / vertical coefficients	On or **Off**	Turn on to map horizontal and vertical coefficients to the same memory. When on and **Load coefficient data at runtime** is also on, writes to the vertical coefficients are ignored. (The choice of read bank remains independent for horizontal and vertical coefficients.)
Vertical Coefficient Data: Memory banks	1–6, Default = **2**	Choose the number of coefficient banks to enable double-buffering, fast coefficient swapping or direct writes.
Vertical Coefficient Data: Filter function	Lanczos 1–12, or Custom, Default = **Lanczos 2**	You can choose from 12 pre-defined Lanczos functions or use the coefficients saved in a custom coefficients file.
Vertical Coefficient Data: Custom coefficient file	used specified	When a custom function is selected, you can browse for a comma-separated value file containing custom coefficients. Use the **Preview coefficients** button to view the current coefficients in a preview window.
Vertical Coefficient Data: Symmetric	On or **Off**	Turn on to save coefficient memory by using symmetric coefficients. When on and **Load coefficient data at runtime** is also on, coefficient writes beyond phases 2 and 1 are ignored.
Horizontal Coefficient Data: Memory banks	1–6, Default = **2**	Choose the number of coefficient banks to enable double-buffering, fast coefficient swapping or direct writes.
Horizontal Coefficient Data: Filter function	Lanczos 1–12, or Custom, Default = **Lanczos 2**	You can choose from 12 pre-defined Lanczos functions or use the coefficients saved in a custom coefficients file.

Table A–11. Scaler Parameter Settings, Coefficients Tab (Part 2 of 2)

Parameter	Value	Description
Horizontal Coefficient Data: Custom coefficient file	used specified	When a custom function is selected, you can browse for a comma-separated value file containing custom coefficients. Use the **Preview coefficients** button to view the current coefficients in a preview window.
Horizontal Coefficient Data: Symmetric	On or **Off**	Turn on to save coefficient memory by using symmetric coefficients. When on and **Load coefficient data at runtime** is also on, coefficient writes beyond phases 2 and 1 are ignored.

Clipper

Table A–12 shows the Clipper parameters.

Table A–12. Clipper Parameter Settings

Parameter	Value	Description
Maximum width	32 to input image width, Default = **1,024**	Specify the maximum width of the clipping rectangle for the input image.
Maximum height	32 to input image height, Default = **768**	Specify the maximum height of the clipping rectangle for the input image.
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Number of color planes in sequence	1–**3**	Choose the number of color planes that are sent in sequence over one data connection. For example, a value of 3 for R'G'B' R'G'B' R'G'B'.
Number of color planes in parallel	**1**–3	Choose the number of color planes in parallel.
Include Avalon-MM interface	On or **Off**	Turn on if you want to specify clipping offsets using the Avalon-MM interface.
Clipping method	Offsets, Rectangle	Choose whether to specify the clipping area as offsets from the edge of the input area or as a fixed rectangle.
Left offset	positive integer, Default = **10**	Specify the x coordinate for the left edge of the clipping rectangle. 0 is the left edge of the input image. *(Note 1)*
Right offset	positive integer, Default = **10**	Specify the x coordinate for the right edge of the clipping rectangle. 0 is the right edge of the input image. *(Note 1)*
Width	positive integer, Default = **10**	Specify the width of the clipping rectangle.
Top offset	positive integer, Default = **10**	Specify the y coordinate for the top edge of the clipping rectangle. 0 is the top edge of the input image. *(Note 2)*
Bottom offset	positive integer, Default = **10**	Specify the y coordinate for the bottom edge of the clipping rectangle. 0 is the bottom edge of the input image. *(Note 2)*
Height	positive integer, Default = **10**	Specify the height of the clipping rectangle.

Notes to Table A–12:

(1) The left and right offset values must be less than or equal to the input image width.

(2) The top and bottom offset values must be less than or equal to the input image height.

Deinterlacer

Table A–13 shows the Deinterlacer MegaCore function parameters.

Table A–13. Deinterlacer Parameter Settings (Part 1 of 2)

Parameter	Value	Description
Maximum image width	32–2,600, Default = **640**	Choose the maximum image width in pixels. The maximum image width is the default width at start-up.
Maximum image height	32–2,600, Default = **480**	Choose the maximum image height in pixels. The maximum image height is the default height at start-up.
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Number of color planes in sequence	1–**3**	Choose the number of color planes that are sent in sequence over one data connection. For example, a value of 3 for R'G'B' R'G'B' R'G'B'.
Number of color planes in parallel	**1**–3	Choose the number of color planes in parallel.
Default initial field	**F0**, F1	Choose a default type for the initial field. The default value is not used if the first field is preceded by an Avalon-ST Control packet.
Deinterlacing Method	**Bob - Scanline Duplication**, Bob - Scanline Interpolation, Weave, Motion Adaptive	Refer to "Deinterlacing Methods" on page 5–21. *(Note 1)*
Frame buffering mode	**No buffering**, Double buffering, Triple buffering with rate conversion	Specifies whether external frame buffers are used. In no buffering mode, data is piped directly from input to output without using external memory. This is possible only with the bob method. Double-buffering routes data via a pair of buffers in external memory. This is required by the weave and motion-adaptive methods, and can ease throughput issues for the bob method. Triple-buffering uses three buffers in external memory and has the advantage over double-buffering that the Deinterlacer can drop or repeat frames, to perform simple frame rate conversion. *(Note 1)*, *(Note 3)*, *(Note 4)*, *(Note 5)*
Output frame rate	**As input frame rate (F0 synchronized)**, As input frame rate (F1 synchronized), As input field rate	Specifies whether to produce a frame out for every field which is input, or a frame output for every frame (pair of fields) input. Each deinterlacing method is defined in terms of its processing of the current field and some number of preceding fields. In the case where a frame is produced only for every two input fields, the current field is either always an F1 field or always an F0 field.
Passthrough mode	On or **Off**	Turn on to propagate progressive frames unchanged. When off, the progressive frames are discarded.
Motion bleed	On or **Off**	Turn on to compare the motion value with thecorresponding motion value for the same location in the previous frame. f it is greater, the new value is kept, but if the new value is less than the stored value, the motion value used is the mean of the two values. This reduces unpleasant flickering artefacts but increases the memory usage and memory bandwidth requirements. *(Note 2)*
Runtime control for locked frame rate conversion	On or **Off**	Turn on to add an Avalon-MM slave interface that synchronizes the input and output frame rates. *(Note 2)*, *(Note 6)*

Table A–13. Deinterlacer Parameter Settings (Part 2 of 2)

Parameter	Value	Description
Runtime control of the motion-adaptive blending	On or **Off**	Turn on to add an Avalon-MM slave interface that controls the behavior of the motion adaptive algorithm at run time. The pixel-based motion value computed by the algorithm can be replaced by a user selected frame-based motion value that varies between the two extremes of being entirely bob or entirely weave. *(Note 4)*, *(Note 6)*
Use separate clocks for the Avalon-MM master interfaces	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 ports width	16, 32, **64**,128, 256	Specifies the width of the Avalon Memory-Mapped (Avalon-MM) ports used to access external memory when double-buffering or triple-buffering is used. *(Note 3)*
Read-only master(s) interface FIFO depth	16–1,024, Default = **64**	Choose the FIFO depth of the read-only Avalon-MM interface.
Read-only master(s) interface burst target	2–256, Default = **32**	Choose the burst target for the read-only Avalon-MM interface.
Write-only master(s) interface FIFO depth	16–1,024, Default = **64**	Choose the FIFO depth of the write-only Avalon-MM interface.
Write-only master(s) interface burst target	8–256, Default = **32**	Choose the burst target for the write-only Avalon-MM interface.
Base address of frame buffers	Any 32-bit value, Default = **0x00000000**	Hexadecimal address of the frame buffers in external memory when buffering is used. *(Note 3)*
Number of packets buffered per field	**1**–32	Specify the number of packets that can be buffered with each field. Older packets are discarded first in case of an overflow. *(Note 5)*
Maximum packet length	**10**–1,024	Choose 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 packets are larger than allowed. *(Note 5)*

Notes to Table A–13:

(1) Either double or triple-buffering mode must be selected before you can select the weave or motion-adaptive deinterlacing methods.
(2) These options are available only when you select Motion Adaptive as the deinterlacing method.
(3) The options to specify the Avalon-MM master ports width and the base address for the frame buffers are available only when you select double or triple-buffering.
(4) The option to synchronize input and output frame rates is only available when double-buffering mode is selected.
(5) The options to control the buffering of non-image data packets are available when you select double or triple-buffering.
(6) You cannot enable both run-time cintrol interfaces at the same time.

Frame Buffer

Table A–14 shows the Frame Buffer parameters.

Table A–14. Frame Buffer Parameter Settings (Part 1 of 2)

Parameter	Value	Description
Maximum image width	32–2,600, Default = **640**	Specify the maximum image width.

Table A–14. Frame Buffer Parameter Settings (Part 2 of 2)

Parameter	Value	Description
Maximum image height	32–2,600, Default = **480**	Specify the maximum image height.
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Number of color planes in sequence	1–**3**	Choose the number of color planes in sequence.
Number of color planes in parallel	**1**–3	Choose the number of color planes in parallel.
Frame dropping	**On** or Off	Turn on to allow frame dropping.
Frame repetition	**On** or Off	Turn on to allow frame repetition.
Runtime control for the writer thread	On or **Off**	Turn on to enable run-time control for the write interfaces.
Runtime control for the reader thread	On or **Off**	Turn on to enable run-time control for the read interfaces.
Use separate clocks for the Avalon-MM master interfaces	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.
External memory port width	16, 32, **64**, 128, 256	Choose the width of the external memory port.
Write-only master interface FIFO depth	16–1,024, Default = **64**	Choose the FIFO depth of the write-only Avalon-MM interface.
Write-only master interface burst target	2–256, Default = **32**	Choose the burst target for the write-only Avalon-MM interface.
Read-only master interface FIFO depth	16–1,024, Default = **64**	Choose the FIFO depth of the read-only Avalon-MM interface.
Read-only master interface burst target	2–256, Default = **32**	Choose the burst target for the read-only Avalon-MM interface.
Base address of frame buffers	Any 32-bit value, Default = **0x00000000**	Hexadecimal address of the frame buffers in external memory.
Number of packets buffered per frame	**1**–32	Specify the maximum number of packets that can be buffered with each frame. Older packets are discarded first in case of an overflow.
Maximum packet length	**10**–1,024	Specify 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 packets are larger than allowed.

Line Buffer Compiler

Table A–15 shows the Line Buffer Compiler parameters.

Table A–15. Line Buffer Compiler Parameter Settings (Part 1 of 2)

Parameter	Value	Description
Line length	1–1,920, Default = **64**	The length of each line buffer in bits.

Table A–15. Line Buffer Compiler Parameter Settings (Part 2 of 2)

Parameter	Value	Description
Line width	1–64, Default = **8**	The width of each line buffer in bits.
Number of lines	1–16, Default = **3**	The number of line buffers required.

Clocked Video Input

Table A–16 shows the Clocked Video Input parameters.

Table A–16. Clocked Video Input Parameter Settings

Parameter	Value	Description
Preset conversion	**DVI 1080p60**, SDI 1080p60, SDI 1080i60, PAL, NTSC	You can choose from a list of preset conversions or use the other fields in the dialog box to set up custom parameter values. If you click **Load values into controls** the dialog box is initialized with values for the selected preset conversion.
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Number of color planes	1–3, Default = **3**	Choose the number of color planes.
Color plane transmission format	Sequence, **Parallel**	Choose whether the color planes are transmitted in sequence or in parallel.
Field order	**Field 0 first**, Field 1 first, Any field first,	Choose the field to sync to first when starting or stopping the output.
Avalon-ST Video Initial / Default Control Packet	**Progressive**, Interlaced	Choose the format to be used when no format can be automatically detected.
Image Width, Progressive / Field 0	32–65,536, Default = **1,920**	Choose the image width to be used when no format can be automatically detected.
Image Width, Field 1	32–65,536, Default = **1,920**	Choose the image width for interlaced field 1when no format can be automatically detected.
Image Height, Progressive / Field 0	32–65,536, Default = **1,080**	Choose the image height to be used when no format can be automatically detected.
Image Height, Field 1	32–65,536, Default = **1,080**	Choose the image height for interlaced field 1when no format can be automatically detected.
Pixel FIFO size	32–(memory limit), Default = **1,920**	Choose the required FIFO depth in pixels (limited by the available on-chip memory).
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.
Sync Signals	Embedded in video, **On separate wires**	Choose whether the synchronization signal is embedded in the video stream or provided on a separate wire.
Allow color planes in sequence input	On or **Off**	Choose whether run-time switching is allowed between sequential and parallel color plane transmission formats. The format is controlled by the `vid_hd_sdn` signal.
Use control port	On or **Off**	Turn on to use the optional stop/go control port.

Clocked Video Output

Table A–17 shows the Clocked Video Output parameters.

Table A–17. Clocked Video Output Parameter Settings (Part 1 of 2)

Parameter	Value	Description
Preset conversion	**DVI 1080p60**, SDI 1080p60, SDI 1080i60, PAL, NTSC	You can choose from a list of preset conversions or use the other fields in the dialog box to set up custom parameter values. If you click **Load values into controls** the dialog box is initialized with values for the selected preset conversion.
Image width / Active pixels	32–65,536, Default = **1,920**	Specify the image width by choosing the number of active pixels.
Image height / Active lines	32–65,536, Default = **1,080**	Specify the image height by choosing the number of active lines.
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Number of color planes	1–3, Default = **3**	Choose the number of color planes.
Color plane transmission format	Sequence, **Parallel**	Choose whether the color planes are transmitted in sequence or in parallel.
Allow output of color planes in sequence	On or **Off**	Choose whether run-time switching is allowed between sequential and parallel color plane transmission formats. The format is controlled by the `ModeXInterlaced` registers.
Interlaced video	On or **Off**	Turn on if you want to use interlaced video. If on, you can set the additional Interlaced and Field 0 Parameters.
Sync signals	Embedded in video, **On separate wires**	Choose whether the synchronization signal is embedded in the video stream or provided on a separate wire. If you choose **Embedded in video**, you can set the active picture line, horizontal blanking, and vertical blanking values. If you choose **On separate wires**, you can set horizontal and vertical values for sync, front porch, and back porch.
Frame / Field 1: Active picture line	0–65,536, Default = **0**	Choose the start of active picture line for Frame/Field 1.
Frame / Field 1: Horizontal blanking	0–65,536, Default = **0**	Choose the size of the horizontal blanking period in pixels for Frame/Field 1.
Frame / Field 1: Vertical blanking	0–65,536, Default = **0**	Choose the size of the vertical blanking period in pixels for Frame/Field 1.
Frame / Field 1: Horizontal sync	1–65,536, Default = **60**	Choose the size of the horizontal sync period in pixels for Frame/Field 1.
Frame / Field 1: Horizontal front porch	1–65,536, Default = **20**	Choose the size of the horizontal front porch period in pixels for Frame/Field 1.
Frame / Field 1: Horizontal back porch	1–65,536, Default = **192**	Choose the size of the horizontal back porch period in pixels for Frame/Field 1.
Frame / Field 1: Vertical sync	0–65,536, Default = **5**	Choose the number of lines in the vertical sync period for Frame/Field 1.
Frame / Field 1: Vertical front porch	0–65,536, Default = **4**	Choose the number of lines in the vertical front porch period for Frame/Field 1.
Frame / Field 1: Vertical back porch	0–65,536, Default = **36**	Choose the number of lines in the vertical back porch period for Frame/Field 1.

Table A–17. Clocked Video Output Parameter Settings (Part 2 of 2)

Parameter	Value	Description
Interlaced and Field 0: F rising edge line	0–65,536, Default = **0**	Choose the line when the rising edge of the field bit occurs for Interlaced and Field 0.
Interlaced and Field 0: F falling edge line	0–65,536, Default = **18**	Choose the line when the rising edge of the vertical blanking bit for Field 0 occurs for Interlaced and Field 0.
Interlaced and Field 0: Vertical blanking rising edge line	0–65,536, Default = **0**	Choose the line when the vertical blanking rising edge occurs for Interlaced and Field 0.
Interlaced and Field 0: Vertical blanking	0–65,536, Default = **0**	Choose the number of lines in the vertical front porch period for Interlaced and Field 0.
Interlaced and Field 0: Vertical sync	0–65,536, Default = **0**	Choose the number of lines in the vertical back porch period for Interlaced and Field 0.
Interlaced and Field 0: Vertical front porch	0–65,536, Default = **0**	Choose the number of lines in the vertical front porch period for Interlaced and Field 0.
Interlaced and Field 0: Vertical back porch	0–65,536, Default = **0**	Choose the number of lines in the vertical back porch period for Interlaced and Field 0.
Pixel FIFO size	32–(memory limit), Default = **1,920**	Choose the required FIFO depth in pixels (limited by the available on-chip memory).
FIFO level at which to start output	0–(memory limit), Default = **0**	Choose 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.
Runtime configurable video modes	1–14, Default = **1**	Choose the number of runtime configurable video output modes that are required when you are using the Avalon-MM control port.

Color Plane Sequencer

Table A–18 shows the Color Plane Sequencer parameters.

Table A–18. Color Plane Sequencer Parameter Settings (Part 1 of 2)

Parameter	Value	Description
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Two pixels per port	On or **Off**	Turn on to enable two pixels on each port.
Color planes in parallel (din0)	1–**3**	Choose the number of color planes in parallel for input port `din0`.
Color planes in sequence (din0)	**1**–4	Choose the number of color planes in sequence for input port `din0`.
Port enabled (din1)	On or **Off**	Turn on to enable input port `din0`.
Color planes in parallel (din1)	1–**3**	Choose the number of color planes in parallel for input port `din1`.
Color planes in sequence (din1)	**1**–4	Choose the number of color planes in sequence for input port `din1`.
Port enabled (dout0)	**On** or Off	Turn on to enable output port `dout0`.
Source non-image packets from port (dout0)	**din0**, din1, din0 and din1	Choose the source port(s) that are enabled for non-image packets for output port `dout0`.
Halve control packet width (dout0)	On or **Off**	Turn on to halve the Avalon-ST Video control packet width for output port `dout0`. *(Note 1)*
Color planes in parallel (dout0)	1–**3**	Choose the number of color planes in parallel for output port `dout0`.

Table A–18. Color Plane Sequencer Parameter Settings (Part 2 of 2)

Parameter	Value	Description
Color planes in sequence (dout0)	**1**–4	Choose the number of color planes in sequence for output port `dout0`.
Port enabled (dout1)	**On** or Off	Turn on to enable output port `dout1`.
Source non-image packets from port (dout1)	**din0**, din1, din0 and din1	Choose the source port used for non-image packets for output port `dout1`.
Halve control packet width (dout1)	On or **Off**	Turn on to halve the Avalon-ST Video control packet width for output port `dout1`. *(Note 1)*
Color planes in parallel (dout1)	1–**3**	Choose the number of color planes in parallel for output port `dout1`.
Color planes in sequence (dout1)	**1**–4	Choose the number of color planes in sequence for output port `dout1`.

Note to Table A–18:

(1) This option can be useful if you want to split a subsampled color plane from a fully sampled color plane. The subsampled color plane can then be processed by other functions as if fully sampled.

Test Pattern Generator

Table A–19 shows the Test Pattern Generator parameters.

Table A–19. Test Pattern Generator Parameter Settings

Parameter	Value	Description
Run-time control of image size	On or **Off**	Turn on to enable run-time control of the image size. When on, the output size parameters control the maximum values.
Maximum image width	32–2,600, Default = **640**	Choose the required output width in pixels.
Maximum image height	32–2,600, Default = **480**	Choose the required output height in pixels.
Bits per pixel per color plane	4–20, Default = **8**	Choose the number of bits per pixel (per color plane).
Color space	RGB or YCbCr, Default = **RGB**	Choose whether to use an R'G'B' or Y'CbCr color space.
Output format	**4:4:4**, 4:2:2, 4:2:0	Choose the format/sampling rate format for the output frames.
Color planes transmission format	**Sequence**, Parallel	This function always outputs three color planes but you can choose whether they are transmitted in sequence or in parallel.
Interlacing	**Progressive output**, Interlaced output (F0 synchronized), Interlaced output (F1 synchronized)	Specifies whether to produce a progressive or an interlaced output stream.
Pattern	Color bars, Uniform background	Choose the standard color bar or a uniform background. If you choose a uniform background, you can specify the individual R'G'B or Y' Cb Cr values depending on the currently selected color space.

Run-Time Control Register Maps

The Color Space Converter, Gamma Corrector, 2D FIR Filter, Alpha Blending Mixer, Scaler, Clocked Video Input, Clocked Video Output, Frame Buffer, and Test Pattern Generator MegaCore functions support run-time control for some of their behavior using a common type of Avalon-MM slave interface. This section describes the control register maps which can be accessed using these interfaces.

For information about the Control and Status registers which are common to these interfaces, refer to "Avalon-MM Slave Interfaces" on page 4–14.

Color Space Converter

Table A–20 describes the control register map for the Color Space Converter.

The width of each register in the Color Space Converter control register map is 32 bits. The coefficient and summand registers use integer, signed 2's complement numbers. 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 MegaCore function, so the registers can be safely updated during the processing of a frame.

Table A–20. Color Space Converter Control Register Map

Address	Register Name	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the Color Space Converter MegaCore function to stop the next time control information is read. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
2	`Coefficient A0`	For details, refer to "Color Space Conversion" on page 5–1.
3	`Coefficient B0`	
4	`Coefficient C0`	
5	`Coefficient A1`	
6	`Coefficient B1`	
7	`Coefficient C1`	
8	`Coefficient A2`	
9	`Coefficient B2`	
10	`Coefficient C2`	
11	`Summand S0`	
12	`Summand S1`	
13	`Summand S2`	

Gamma Corrector

The Gamma Corrector can have up to three Avalon-MM slave interfaces. There is a separate slave interface for each channel in parallel. Table A–21, Table A–22 and Table A–23 on page A–17 describe the control register maps for these interfaces.

The control registers are read continuously during the operation of the MegaCore function, so making a change to part of the Gamma look-up table during the processing of a frame always has immediate effect. To synchronize changes to frame boundaries, follow the procedure which is described in "Avalon-MM Slave Interfaces" on page 4–14.

The width of each register in the Gamma Corrector control register map is always equal to the value of the *Bits per pixel per color plane* parameter selected in the MegaWizard interface.

Table A–21. Gamma Corrector Control Register Map: Interface 0

Address	Register Name	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the Gamma Corrector MegaCore function to stop the next time control information is read. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
2 to 2^N +1 where N is the number of bits per color plane.	`Gamma Look-Up Table`	These registers contain a look-up table that is used to apply gamma correction to video data. An input intensity value of x is gamma corrected by replacing it with the contents of the (x+1)th entry in the look-up table. Changing the values of these registers has an immediate effect on the behavior of the MegaCore function. To ensure that gamma look-up values do not change during processing of a video frame, use the `Go` bit to stop the MegaCore function while the table is changed.

Table A–22. Gamma Corrector Control Register Map: Interface 1

Address	Register Name	Description
0	`Unused`	This register is not used
1	`Unused`	This register is not used
2 to 2^N +1 where N is the number of bits per color plane.	`Gamma Look-Up Table`	These registers contain a look-up table that is used to apply gamma correction to video data. An input intensity value of x is gamma corrected by replacing it with the contents of the (x+1)th entry in the look-up table. Changing the values of these registers has an immediate effect on the behavior of the MegaCore function. To ensure that gamma look-up values do not change during processing of a video frame, use the `Go` bit in Interface 0 to stop the MegaCore function while the table is changed.

Table A–23. Gamma Corrector Control Register Map: Interface 2

Address	Register Name	Description
0	`Unused`	This register is not used
1	`Unused`	This register is not used
2 to 2^N +1 where N is the number of bits per color plane.	`Gamma Look-Up Table`	These registers contain a look-up table that is used to apply gamma correction to video data. An input intensity value of x is gamma corrected by replacing it with the contents of the (x+1)th entry in the look-up table. Changing the values of these registers has an immediate effect on the behavior of the MegaCore function. To ensure that gamma look-up values do not change during processing of a video frame, use the `Go` bit in Interface 0 to stop the MegaCore function while the table is changed.

2D FIR Filter

Table A–24 describes the control register map for the 2D FIR Filter.

The width of each register in the 2D FIR Filter control register map is 32 bits. The coefficient registers use integer, signed 2's complement numbers. 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 MegaCore function, so the registers can be safely updated during the processing of a frame.

Table A–24. 2D FIR Filter Control Register Map

Address	Register Name	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the 2D FIR Filter MegaCore function to stop the next time control information is read. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
2	`Coefficient 0`	The coefficient at the top left (origin) of the filter kernel.
3	`Coefficient 1`	The coefficient at the origin across to the right by one.
4	`Coefficient 2`	The coefficient at the origin across to the right by two.
n	`Coefficient` *n*	The coefficient at position: ■ Row (where 0 is the top row of the kernel) is the integer value via the truncation of (*n*–2) / (filter kernel width) ■ Column (where 0 is the far left row of the kernel) is the remainder of (*n*–2) / (filter kernel width)

Alpha Blending Mixer

Table A–25 describes the Alpha Blending Mixer control register map.

The width of each register in the Alpha Blending Mixer control register map is 16 bits. The control data is read once at the start of each frame and is buffered inside the MegaCore function, so the registers may be safely updated during the processing of a frame.

Table A–25. Alpha Blending Mixer Control Register Map (Part 1 of 2)

Address	Register(s)	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the Alpha Blending Mixer MegaCore function to stop the next time control information is read. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
2	`Layer 1 X`	Offset in pixels from the left edge of the background layer to the left edge of layer 1. *(Note 1)*
3	`Layer 1 Y`	Offset in pixels from the top edge of the background layer to the top edge of layer 1. *(Note 1)*

Table A–25. Alpha Blending Mixer Control Register Map (Part 2 of 2)

Address	Register(s)	Description
4	`Layer 1 Active`	Layer 1 is displayed if this control register is set to 1. Data in the input stream is consumed but not displayed if this control register is set to 2, Avalon-ST packets of type 2 to 14 are still propagated as usual. Data from the input stream is not pulled out if this control register is set to 0. *(Note 1)*, *(Note 2)*.
5	`Layer 2 X`	.... *(Note 3)*

Note to Table A–25:

(1) The value of this register is checked at the start of each frame. If the register is changed during the processing of a video frame, the change does not take effect until the start of the next frame.

(2) For efficiency reasons, the Video and Image Processing Suite MegaCore functions 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.

(3) The rows in the table are repeated in ascending order for each layer from 1 to the foreground layer.

Scaler

Table A–26 describes the Scaler control register map.

The control data is read once at the start of each frame and is buffered inside the MegaCore function, so the registers may be safely updated during the processing of a frame. Note that all Scaler registers are write-only except at address 1.

Table A–26. Scaler Control Register Map (Part 1 of 2)

Address	Register	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0, causes the Scaler to stop the next time that control information is read. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. The Scaler MegaCore function sets this address to 0 between frames. It is set to 1 while the MegaCore function is processing data and cannot be stopped. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
2	`Output Width`	The width of the output frames in pixels. *(Note 1)*
3	`Output Height`	The height of the output frames in pixels. *(Note 1)*
4	`Horizontal Coefficient Bank Write Address`	Specifies which memory bank horizontal coefficient writes from the Avalon-MM interface are made into.
5	`Horizontal Coefficient Bank Read Address`	Specifies which memory bank is used for horizontal coefficient reads during data processing.
6	`Vertical Coefficient Bank Write Address`	Specifies which memory bank vertical coefficient writes from the Avalon-MM interface are made into. *(Note 2)*
7	`Vertical Coefficient Bank Read Address`	Specifies which memory bank is used for vertical coefficient reads during data processing
8 to 7+N_h	`Horizontal Tap Data`	Specifies values for the horizontal coefficients at a particular phase. Write these values first, then the `Horizontal Phase` to commit the write.
8+N_h	`Horizontal Phase`	Specifies which phase the `Horizontal Tap Data` applies to. Writing to this location, commits the writing of tap data. This write must be made even if the phase value does not change between successive sets of tap data.

Table A–26. Scaler Control Register Map (Part 2 of 2)

Address	Register	Description
$9+N_h$ to $8+N_h+N_h+N_v$	`Vertical Tap Data`	Specifies values for the vertical coefficients at a particular phase. Write these values first, then the `Vertical Phase` to commit the write. *(Note 2)*
$9+N_h+N_v$	`Vertical Phase`	Specifies which phase the `Vertical Tap Data` applies to. Writing to this location, commits the writing of tap data. This write must be made even if the phase value does not change between successive sets of tap data. *(Note 2)*

Note to Table A–26:

(1) Value can be from 32 to the maximum specified in the MegaWizard interface.

(2) If **Share horizontal/vertical coefficients** is selected in the MegaWizard interface, this location is not used.

Table A–27 shows an example of the sequence of writes to the horizontal coefficient data for an instance of the Scaler MegaCore function with four taps and eight phases.

Table A–27. Example of Using the Scaler Control Registers

Address	Value	Purpose
8	0	Setting up Tap 0 for Phase 0.
9	128	Setting up Tap 1 for Phase 0.
10	0	Setting up Tap 2 for Phase 0.
11	0	Setting up Tap 3 for Phase 0.
12	0	Commit the writes to Phase 0.
8	–8	Setting up Tap 0 for Phase 1.
9	124	Setting up Tap 1 for Phase 1.
10	13	Setting up Tap 2 for Phase 1.
11	–1	Setting up Tap 3 for Phase 1.
12	0	Commit the writes to Phase 1.
...	...	...
8	–1	Setting up Tap 0 for Phase 7.
9	13	Setting up Tap 1 for Phase 7.
10	124	Setting up Tap 2 for Phase 7.
11	–8	Setting up Tap 3 for Phase 7.
12	0	Commit the writes to Phase 7.

Clipper

Table A–28 describes the Clipper control register map.

The control data is read once at the start of each frame and is buffered inside the MegaCore function, so the registers can be safely updated during the processing of a frame. Note that all Clipper registers are write-only except at address 1.

Table A–28. Clipper Control Register Map

Address	Register	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the Clipper MegaCore function to stop the next time control information is read. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. The Clipper MegaCore function sets this address to 0 between frames. It is set to 1 while the MegaCore function is processing data and cannot be stopped. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
2	`Left Offset`	The left offset, in pixels, of the clipping window/rectangle. *(Note 1)*
3	`Right Offset or Width`	In clipping window mode, the right offset of the window. In clipping rectangle mode, the width of the rectangle. *(Note 1)*
4	`Top Offset`	The top offset, in pixels, of the clipping window/rectangle. *(Note 2)*
5	`Bottom Offset or Height`	In clipping window mode, the bottom offset of the window. In clipping rectangle mode, the height of the rectangle. *(Note 2)*

Note to Table A–28:

(1) The left and right offset values must be less than or equal to the input image width.

(2) The top and bottom offset values must be less than or equal to the input image height.

Deinterlacer

An run-time control interface can be attached to the Deinterlacer that you can use to override the default behavior of the motion-adaptive algorithm or to synchronize the input and output frame rates. However, it is not possible to enable both of these interface simultaneously.

Table A–29 describes the control register map that controls the motion-adaptive algorithm at run time. The control data is read once and registered before outputting a frame. It can be safely updated during the processing of a frame.

Table A–29. Deinterlacer Control Register Map (Part 1 of 2)

Address	Register	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the Deinterlacer MegaCore function to stop before control information is read and before outputting a frame. While stopped, the Deinterlacer may continue to receive and drop frames at its input if triple-buffering is enabled. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
2	`Motion value override`	Write-only register. The zeroth bit of this register should be set to 1 to override the per-pixel motion value computed by the deinterlacing algorithm with a user specified value. This register cannot be read.

Table A–29. Deinterlacer Control Register Map (Part 2 of 2)

Address	Register	Description
3	`Blending coefficient`	Write-only register. The 16-bit value that overrides the motion value computed by the deinterlacing algorithm. This value can vary between 0 (weaving) to 65535 (bobbing). The register cannot be read.

Table A–29 describes the control register map that synchronizes the input and output frame rates. The control data is read and registered when receiving the image data header that signals new frame. It can be safely updated during the processing of a frame.

Table A–30. Deinterlacer Control Register Map

Address	Register	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the Deinterlacer MegaCore function to stop before control information is read and before outputting a frame. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
2	`Input frame rate`	Write-only register. An 8-bit integer value for the input frame rate This register cannot be read. *(Note 1)*
3	`Output frame rate`	Write-only register. An 8-bit integer value for the output frame rate. The register cannot be read. *(Note 1)*

Note to Table A–30:

(1) The behavior of the rate conversion algorithm is not directly affected by a particular choice of input and output rates but only by their ratio. 23.976 -> 29.970 is equivalent to 24 -> 30.

Frame Buffer

A run-time control can be attached either to the writer component or to the reader component of the Frame Buffer MegaCore function but not to both. The width of each register is 16 bits.

Table A–31 describes the Frame Buffer control register map for the writer component.

Table A–31. Frame Buffer Control Register Map for the Writer Component

Address	Register(s)	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the Frame Buffer MegaCore function to stop the next time control information is read. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
2	`Frame Counter`	Read-only register updated at the end of each frame processed by the writer. The counter is incremented if the frame is not dropped and passed to the reader component.
3	`Drop Counter`	Read-only register updated at the end of each frame processed by the writer. The counter is incremented if the frame is dropped.

Table A–32 describes the Frame Buffer control register map for the reader component.

Table A–32. Frame Buffer Control Register Map for the Reader Component

Address	Register(s)	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the Frame Buffer MegaCore function to stop the next time control information is read. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. Refer to "Avalon-MM Slave Interfaces" on page 4–14 for full details.
2	`Frame Counter`	Read-only register updated at the end of each frame processed by the reader. The counter is incremented if the frame is not repeated.
3	`Repeat Counter`	Read-only register updated at the end of each frame processed by the reader. The counter is incremented if the frame is about to be repeated.

Clocked Video Input

Table A–33 describes the Clocked Video Input control register map.

The width of each register is 16 bits.

Table A–33. Clocked Video Input Control Register Map (Part 1 of 2)

Address	Register	Description
0	`Control`	The zeroth bit of this register is the `Go` bit: ■ Setting this bit to 1 causes the Clocked Video Input MegaCore function to start data output on the next video frame boundary. Refer to "Control Port" on page 5–32 for full details. Bits 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 stable video interrupt.
1	`Status`	The zeroth bit of this register is the `Status` bit: ■ Data is being output by the Clocked Video Input MegaCore function when this bit is asserted. Refer to "Control Port" on page 5–32 for full details. Bits 2 and 1 of the `Status` register are the interrupt status bits: ■ When bit 1 is asserted, the status update interrupt has triggered. ■ When bit 2 is asserted, the stable video interrupt has triggered. ■ The interrupts stay asserted until a write of 1 is performed to these bits. Bits 6, 5, 4, and 3 are the resolution valid bits: ■ When bit 3 is asserted, the `F0SampleCount` register is valid. ■ When bit 4 is asserted, the `F0LineCount` register is valid. ■ When bit 5 is asserted the `F1SampleCount` register is valid. ■ When bit 6 is asserted the `F1LineCount` register is valid. 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 resolution for two frames, in the case of progressive video or four fields, in the case of interlaced video.

Table A–33. Clocked Video Input Control Register Map (Part 2 of 2)

Address	Register	Description
1	Status	Bit 9 is the overflow sticky bit: ■ When asserted the input FIFO has overflowed. The overflow sticky bit stays asserted until a write of is performed to this bit.
2	UsedW	The used words level of the input FIFO.
3	F0SampleCount	The detected sample count of the video streams F0 field.
4	F0LineCount	The detected line count of the video streams F0 field.
5	F1SampleCount	The detected sample count of the video streams F1 field.
6	F1LineCount	The detected line count of the video streams F1 field.

Clocked Video Output

Table A–34 describes the Clocked Video Output control register map.

The width of each register is 16 bits.

Table A–34. Clocked Video Output Control Register Map (Part 1 of 2)

Address	Register	Description
0	Control	The zeroth bit of this register is the Go bit: ■ Setting this bit to 1 causes the Clocked Video Output MegaCore function to start video data output. Refer to "Control Port" on page 5–37 for full details. Bit 1 of the Control register is the interrupt mask: ■ Setting bit 1 to 1, masks the status update interrupt.
1	Status	The zeroth bit of this register is the Status bit: ■ Data is being output by the Clocked Video Output MegaCore function when this bit is asserted. Refer to "Control Port" on page 5–37 for full details. Bit 1 is the interrupt status bit: ■ When bit 1 is asserted, the status update interrupt has triggered. The interrupt stays asserted until a write of 1 is performed to this bit. 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 1is written to this bit.
2	UsedW	The used words level of the output FIFO.
3	VidModeMatch	One hot register that indicates the video mode that is selected.
4	Mode1Interlaced	Video Mode 1 Interlaced. Setting bit 0 to a 1 causes the mode to output interlaced video. All other bits are unused. In run-time switching of color plane transmission formats mode only, setting bit 1 to a 1 causes the mode to output sequential video; setting it to a 0 outputs parallel video.
5	Mode1F0SampleCount	Video Mode 1 Field 0/Progressive sample count. Specifies the active picture width of the field.
6	Mode1F0LineCount	Video Mode 1 Field 0/Progressive line count. Specifies the active picture height of the field.
7	Mode1F1SampleCount	(Interlaced Video Only.) Video Mode 1 Field 1 sample count. Specifies the active picture width of the field.

Table A–34. Clocked Video Output Control Register Map (Part 2 of 2)

Address	Register	Description
8	Mode1F1LineCount	(Interlaced Video Only.) Video Mode 1 Field 1 line count. Specifies the active picture height of the field.
9	Mode1HFrontPorch	Video Mode 1 Horizontal Front Porch. Specifies the length of the horizontal front porch in samples.
10	Mode1HSyncLength	Video Mode 1 Horizontal Sync Length. Specifies the length of the horizontal sync length in samples.
11	Mode1HBlanking	Video Mode 1 Horizontal Blanking Period. Specifies the length of the horizontal blanking period in samples.
12	Mode1VFrontPorch	Video Mode 1 Vertical Front Porch. Specifies the length of the vertical front porch in lines.
13	Mode1VSyncLength	Video Mode 1 Vertical Sync Length. Specifies the length of the vertical sync length in lines.
14	Mode1VBlanking	Video Mode 1 Vertical Blanking Period. Specifies the length of the vertical blanking period in lines.
15	Mode1F0VFrontPorch	(Interlaced Video Only.) Video Mode 1 Field 0 Vertical Front Porch. Specifies the length of the vertical front porch in lines.
16	Mode1F0VSyncLength	(Interlaced Video Only.) Video Mode 1 Field 0 Vertical Sync Length. Specifies the length of the vertical sync length in lines.
17	Mode1F0VBlanking	(Interlaced Video Only.) Video Mode 1 Field 0 Vertical Blanking Period. Specifies the length of the vertical blanking period in lines.
18	Mode1APLine	Video Mode 1 Active Picture Line. Specifies the line number given to the first line of active picture.
19	Mode1F0VRising	Video Mode 1 Field 0 Vertical Blanking Rising Edge. Specifies the line number given to the start of field 0's vertical blanking.
20	Mode1FRising	Video Mode 1 Field Rising Edge. Specifies the line number given to the end of Field 0 and the start of Field 1.
21	Mode1FFalling	Video Mode 1 Field Falling Edge. Specifies the line number given to the end of Field 0 and the start of Field 1.
22	Mode1Valid	Video Mode 1 Valid. Set to indicate that this mode is valid and can be used for video output.
23	Mode2Interlaced	...
24	... *(Note 1)*	...

Note to Table A–34:

(1) The rows in the table are repeated in ascending order for each video mode.

Test Pattern Generator

The width of each register in the Test Pattern Generator control register map is 16 bits. The control data is read once at the start of each frame and is buffered inside the MegaCore function, so that the registers can be safely updated during the processing of a frame.

After control data has been read, the Test Pattern Generator MegaCore function outputs 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 A–35 describes the Test Pattern Generator control register map.

Table A–35. Test Pattern Generator Control Register Map

Address	Register(s)	Description
0	`Control`	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this bit to 0 causes the Test Pattern Generator MegaCore function to stop before control information is read. Refer to "Generation of Avalon-ST Video Control Packets and Run-Time Control" on page 5–45 for full details.
1	`Status`	The zeroth bit of this register is the `Status` bit, all other bits are unused. The Test Pattern Generator MegaCore function sets this address to 0 between frames. It is set to 1 while the MegaCore function is producing data and cannot be stopped. Refer to "Generation of Avalon-ST Video Control Packets and Run-Time Control" on page 5–45 for full details.
2	`Output Width`	The width of the output frames in pixels. *(Note 1)*
3	`Output Height`	The height of the output frames in pixels. *(Note 1)*

Note to Table A–35:

(1) Value can be from 32 to the maximum specified in the MegaWizard interface.

Signals

Table A–36 to Table A–46 list the input and output signals for the Video and Image Processing Suite MegaCore functions.

Color Space Converter

Table A–36 shows the input and output signals for the Color Space Converter MegaCore function.

Table A–36. Color Space Converter Signals (Part 1 of 2)

Signal	Direction	Description
`clock`	In	The main system clock. The MegaCore function operates on the rising edge of the `clock` signal.
`reset`	In	The MegaCore function is asynchronously reset when `reset` is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the `clock` signal.
`din_data`	In	Avalon-ST `data` bus for port `din`. Pixel data is transferred into the MegaCore function over this bus.
`din_endofpacket`	In	Avalon-ST `endofpacket` signal for port `din`. This signal marks the end of an Avalon-ST packet.
`din_ready`	Out	Avalon-ST `ready` signal for port `din`. This signal indicates when the MegaCore function is ready to receive data.
`din_startofpacket`	In	Avalon-ST `startofpacket` signal for port `din`. This signal marks the start of an Avalon-ST packet.
`din_valid`	In	Avalon-ST `valid` signal for port `din`. This signal identifies the cycles when the port should input data.
`dout_data`	Out	Avalon-ST `data` bus for port `dout`. Pixel data is transferred out of the MegaCore function over this bus.

Table A–36. Color Space Converter Signals (Part 2 of 2)

Signal	Direction	Description
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.

Chroma Resampler

Table A–37 shows the input and output signals for the Chroma Resampler MegaCore function.

Table A–37. Chroma Resampler Signals

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
din_data	In	Avalon-ST data bus for port din. Pixel data is transferred into the MegaCore function over this bus.
din_endofpacket	In	Avalon-ST endofpacket signal for port din. This signal marks the end of an Avalon-ST packet.
din_ready	Out	Avalon-ST ready signal for port din. This signal indicates when the MegaCore function is ready to receive data.
din_startofpacket	In	Avalon-ST startofpacket signal for port din. This signal marks the start of an Avalon-ST packet.
din_valid	In	Avalon-ST valid signal for port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.

Gamma Corrector

Table A–38 shows the input and output signals for the Gamma Corrector MegaCore function.

Table A–38. Gamma Corrector Signals

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
din_data	In	Avalon-ST data bus for port din. Pixel data is transferred into the MegaCore function over this bus.
din_endofpacket	In	Avalon-ST endofpacket signal for port din. This signal marks the end of an Avalon-ST packet.
din_ready	Out	Avalon-ST ready signal for port din. This signal indicates when the MegaCore function is ready to receive data.
din_startofpacket	In	Avalon-ST startofpacket signal for port din. This signal marks the start of an Avalon-ST packet.
din_valid	In	Avalon-ST valid signal for port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.
gamma_lut_av_address	In	Avalon-MM address bus of slave port gamma_lut. Specifies a word offset into the slave address space.
gamma_lut_av_chipselect	In	Avalon-MM chipselect signal of slave port gamma_lut. The gamma_lut port ignores all other signals unless this signal is asserted.
gamma_lut_av_readdata	Out	Avalon-MM readdata bus of slave port gamma_lut. These output lines are used for read transfers.
gamma_lut_av_write	In	Avalon-MM write signal of slave port gamma_lut. When this signal is asserted, the gamma_lut port accepts new data from the writedata bus.
gamma_lut_av_writedata	In	Avalon-MM writedata bus of slave port gamma_lut. These input lines are used for write transfers.

2D FIR Filter

Table A–39 shows the input and output signals for the 2D FIR Filter MegaCore function.

Table A–39. 2D FIR Filter Signals

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
din_data	In	Avalon-ST data bus for port din. Pixel data is transferred into the MegaCore function over this bus.
din_endofpacket	In	Avalon-ST endofpacket signal for port din. This signal marks the end of an Avalon-ST packet.
din_ready	Out	Avalon-ST ready signal for port din. This signal indicates when the MegaCore function is ready to receive data.
din_startofpacket	In	Avalon-ST startofpacket signal for port din. This signal marks the start of an Avalon-ST packet.
din_valid	In	Avalon-ST valid signal for port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.

2D Median Filter

Table A–40 shows the input and output signals for the 2D Median Filter MegaCore function.

Table A–40. 2D Median Filter Signals (Part 1 of 2)

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
din_data	In	Avalon-ST data bus for port din. Pixel data is transferred into the MegaCore function over this bus.
din_endofpacket	In	Avalon-ST endofpacket signal for port din. This signal marks the end of an Avalon-ST packet.

Table A–40. 2D Median Filter Signals (Part 2 of 2)

Signal	Direction	Description
din_ready	Out	Avalon-ST ready signal for port din. This signal indicates when the MegaCore function is ready to receive data.
din_startofpacket	In	Avalon-ST startofpacket signal for port din. This signal marks the start of an Avalon-ST packet.
din_valid	In	Avalon-ST valid signal for port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.

Alpha Blending Mixer

Table A–41 shows the input and output signals for the Alpha Blending Mixer MegaCore function.

Table A–41. Alpha Blending Mixer Signals (Part 1 of 2)

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
alpha_in_N_data	In	Avalon-ST alpha data bus for port din for layer *N*. Pixel data is transferred into the MegaCore function over this bus. *(Note 1)*
alpha_in_N_endofpacket	In	Avalon-ST endofpacket signal for port alpha_in_N. This signal marks the end of an Avalon-ST packet. *(Note 1)*
alpha_in_N_ready	Out	Avalon-ST alpha ready signal for port din for layer *N*. This signal indicates when the MegaCore function is ready to receive data. *(Note 1)*
alpha_in_N_startofpacket	In	Avalon-ST startofpacket signal for port alpha_in_N. This signal marks the start of an Avalon-ST packet. *(Note 1)*
alpha_in_N_valid	In	Avalon-ST alpha valid signal for port din for layer *N*. This signal identifies the cycles when the port should input data. *(Note 1)*
din_N_data	In	Avalon-ST data bus for port din for layer *N*. Pixel data is transferred into the MegaCore function over this bus.
din_N_endofpacket	In	Avalon-ST endofpacket signal for port din_N. This signal marks the end of an Avalon-ST packet.
din_N_ready	Out	Avalon-ST ready signal for port din for layer *N*. This signal indicates when the MegaCore function is ready to receive data.

Table A–41. Alpha Blending Mixer Signals (Part 2 of 2)

Signal	Direction	Description
din_N_startofpacket	In	Avalon-ST startofpacket signal for port din_N. This signal marks the start of an Avalon-ST packet.
din_N_valid	In	Avalon-ST valid signal for port din for layer *N*. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.
control_av_address	In	Avalon-MM address bus of slave port mix_control. Specifies a word offset into the slave address space.
control_av_chipselect	In	Avalon-MM chipselect signal of slave port mix_control. The gamma_lut port ignores all other signals unless this signal is asserted.
control_av_readdata	Out	Avalon-MM readdata bus of slave port mix_control. These output lines are used for read transfers.
control_av_write	In	Avalon-MM write signal of slave port mix_control. When this signal is asserted, the mix_control port accepts new data from the writedata bus.
control_av_writedata	In	Avalon-MM writedata bus of slave port mix_control. These input lines are used for write transfers.

Note to Table A–41

(1) These ports are present only if **Alpha blending** is on in the MegaWizard interface. Note that alpha channel ports are created for layer zero even though no alpha mixing is possible for layer zero (the background layer). These ports are ignored and can safely be left unconnected.

Scaler

Table A–42 shows the input and output signals for the Scaler MegaCore function.

Table A–42. Scaler Signals (Part 1 of 2)

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
din_data	In	Avalon-ST data bus for port din. Pixel data is transferred into the MegaCore function over this bus.
din_endofpacket	In	Avalon-ST endofpacket signal for port din. This signal marks the end of an Avalon-ST packet.
din_ready	Out	Avalon-ST ready signal for port din. This signal indicates when the MegaCore function is ready to receive data.

Table A–42. Scaler Signals (Part 2 of 2)

Signal	Direction	Description
din_startofpacket	In	Avalon-ST startofpacket signal for port din. This signal marks the start of an Avalon-ST packet.
din_valid	In	Avalon-ST valid signal for port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.
control_av_address	In	Avalon-MM address bus of slave port control. Specifies a word offset into the slave address space. *(Note 1)*
control_av_chipselect	In	Avalon-MM chipselect signal of slave port control. The control port ignores all other signals unless this signal is asserted. *(Note 1)*
control_av_readdata	Out	Avalon-MM readdata bus of slave port control. These output lines are used for read transfers. *(Note 1)*
control_av_waitrequest	Out	Avalon-MM waitrequest signal of slave port control. *(Note 1)*
control_av_write	In	Avalon-MM write signal of slave port control. When this signal is asserted, the control port accepts new data from the writedata bus. *(Note 1)*
control_av_writedata	In	Avalon-MM writedata bus of slave port control. These input lines are used for write transfers. *(Note 1)*

Note to Table A–42

(1) These ports are present only if **Run-time control of image size** is on in the MegaWizard interface.

Clipper

Table A–43 shows the input and output signals for the Clipper MegaCore function.

Table A–43. Clipper Signals (Part 1 of 2)

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
din_data	In	Avalon-ST data bus for port din. Pixel data is transferred into the MegaCore function over this bus.
din_endofpacket	In	Avalon-ST endofpacket signal for port din. This signal marks the end of an Avalon-ST packet.

Table A–43. Clipper Signals (Part 2 of 2)

Signal	Direction	Description
din_ready	Out	Avalon-ST ready signal for port din. This signal indicates when the MegaCore function is ready to receive data.
din_startofpacket	In	Avalon-ST startofpacket signal for port din. This signal marks the start of an Avalon-ST packet.
din_valid	In	Avalon-ST valid signal for port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.
control_av_address	In	Avalon-MM address bus of slave port control. Specifies a word offset into the slave address space. *(Note 1)*
control_av_chipselect	In	Avalon-MM chipselect signal of slave port control. The control port ignores all other signals unless this signal is asserted. *(Note 1)*
control_av_readdata	Out	Avalon-MM readdata bus of slave port control. These output lines are used for read transfers. *(Note 1)*
control_av_waitrequest	Out	Avalon-MM waitrequest signal of slave port control. *(Note 1)*
control_av_write	In	Avalon-MM write signal of slave port control. When this signal is asserted, the control port accepts new data from the writedata bus. *(Note 1)*
control_av_writedata	In	Avalon-MM writedata bus of slave port control. These input lines are used for write transfers. *(Note 1)*

Note to Table A–43

(1) These ports are present only if **Include Avalon-MM interface** is on in the MegaWizard interface.

Deinterlacer

Table A–44 shows the input and output signals for the Deinterlacer MegaCore function.

Table A–44. Deinterlacer Signals (Part 1 of 3)

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
din_data	In	Avalon-ST data bus for port din. Pixel data is transferred into the MegaCore function over this bus.

Table A–44. Deinterlacer Signals (Part 2 of 3)

Signal	Direction	Description
din_endofpacket	In	Avalon-ST endofpacket signal for port din. This signal marks the end of an Avalon-ST packet.
din_ready	Out	Avalon-ST ready signal for port din. This signal indicates when the MegaCore function is ready to receive data.
din_startofpacket	In	Avalon-ST startofpacket signal for port din. This signal marks the start of an Avalon-ST packet.
din_valid	In	Avalon-ST valid signal for port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.
read_master_N_av_address	Out	Avalon-MM address bus of read_master_N port. Specifies a byte address in the Avalon-MM address space. *(Note 1)*, *(2)*, *(3)*
read_master_N_av_burstcount	Out	Avalon-MM burstcount signal of read_master_N port. Specifies the number of transfers in each burst. *(Note 1)*, *(2)*, *(3)*
read_master_N_av_clock	In	The clock signal for the read_master_N port. The interface operates on the rising edge of the clock signal. *(Note 1)*, *(2)*, *(3)*, *(4)*
read_master_N_av_read	Out	Avalon-MM read signal of read_master_N port. Asserted to indicate read requests from the master to the system interconnect fabric. *(Note 1)*, *(2)*, *(3)*
read_master_N_av_readdata	In	Avalon-MM readdata bus of read_master port. These input lines carry data for read transfers. *(Note 1)*, *(2)*, *(3)*
read_master_N_av_readdatavalid	In	Avalon-MM readdatavalid signal of read_master_N port. This signal is asserted by the system interconnect fabric when requested read data has arrived. *(Note 1)*, *(2)*, *(3)*
read_master_N_av_reset	In	The reset signal for the read_master_N port. The interface is asynchronously reset when reset is asserted high and must be de-asserted synchronously with respect to the rising edge of the clock signal. *(Note 1)*, *(2)*, *(3)*, *(4)*
read_master_N_av_waitrequest	In	Avalon-MM waitrequest signal of read_master_N port. Asserted by the system interconnect fabric to cause the master port to wait. *(Note 1)*, *(2)*, *(3)*
write_master_av_address	Out	Avalon-MM address bus of write_master port. Specifies a byte address in the Avalon-MM address space. *(Note 1)*, *(3)*

Table A–44. Deinterlacer Signals (Part 3 of 3)

Signal	Direction	Description
write_master_av_burstcount	Out	Avalon-MM burstcount signal of write_master port. Specifies the number of transfers in each burst. *(Note 1)*, *(2)*, *(3)*
write_master_av_clock	In	The clock signal for the write_master port. The interface operates on the rising edge of the clock signal. *(Note 1)*, *(3)*, *(4)*
write_master_av_reset	In	The reset signal for the write_master port. The interface is asynchronously reset when reset is asserted high and must be de-asserted synchronously with respect to the rising edge of the clock signal. *(Note 1)*, *(3)*, *(4)*
write_master_av_waitrequest	In	Avalon-MM waitrequest signal of write_master port. Asserted by the system interconnect fabric to cause the master port to wait. *(Note 1)*, *(3)*
write_master_av_write	Out	Avalon-MM write signal of write_master port. Asserted to indicate write requests from the master to the system interconnect fabric. *(Note 1)*, *(3)*
write_master_av_writedata	Out	Avalon-MM writedata bus of write_master port. These output lines carry data for write transfers. *(Note 1)*, *(3)*

Note to Table A–44:

(1) The write_master_* and read_master_* signals are present only when buffering is used.

(2) When the motion-adaptive algorithm is selected, two read master interfaces are used.

(3) When the motion-adaptive algorithm is selected, one additional read master (motion_read_master_*) and one additional write master (motion_write_master_*) are used to read and update motion values.

(4) Additional clock and reset signals are available when **Use separate clocks for the Avalon-MM master interfaces** is on in the MegaWizard interface.

Frame Buffer

Table A–45 shows the input and output signals for the Frame Buffer MegaCore function.

Table A–45. Frame Buffer Signals (Part 1 of 3)

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
din_data	In	Avalon-ST data bus for port din. Pixel data is transferred into the MegaCore function over this bus.
din_endofpacket	In	Avalon-ST endofpacket signal for port din. This signal marks the end of an Avalon-ST packet.
din_ready	Out	Avalon-ST ready signal for port din. This signal indicates when the MegaCore function is ready to receive data.
din_startofpacket	In	Avalon-ST startofpacket signal for port din. This signal marks the start of an Avalon-ST packet.

Table A–45. Frame Buffer Signals (Part 2 of 3)

Signal	Direction	Description
din_valid	In	Avalon-ST valid signal for port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function is outputs data.
read_master_av_address	Out	Avalon-MM address bus of read_master port. Specifies a byte address in the Avalon-MM address space.
read_master_av_burstcount	Out	Avalon-MM burstcount signal of read_master port. Specifies the number of transfers in each burst.
read_master_av_clock	In	The clock signal for the read_master port. The interface operates on the rising edge of the clock signal. *(Note 1)*
read_master_av_read	Out	Avalon-MM read signal of read_master port. Asserted to indicate read requests from the master to the system interconnect fabric.
read_master_av_readdata	In	Avalon-MM readdata bus of read_master port. These input lines carry data for read transfers.
read_master_av_readdatavalid	In	Avalon-MM readdatavalid signal of read_master port. This signal is asserted by the system interconnect fabric when requested read data has arrived.
read_master_av_reset	In	The reset signal for the read_master port. The interface is asynchronously reset when reset is asserted high and must be de-asserted synchronously with respect to the rising edge of the clock signal. *(Note 1)*
read_master_av_waitrequest	In	Avalon-MM waitrequest signal of read_master port. Asserted by the system interconnect fabric to cause the master port to wait.
reader_control_av_chipselect	In	Avalon-MM chipselect signal of reader_control slave port. The reader_control port ignores all other signals unless this signal is asserted.
reader_control_av_readdata	Out	Avalon-MM readdata bus of reader_control slave port. These output lines are used for read transfers.
reader_control_av_write	In	Avalon-MM write signal of reader_control slave port. When this signal is asserted, reader_control accepts new data from the writedata bus.
reader_control_av_writedata	In	Avalon-MM writedata bus of reader_control slave port These input lines are used for write transfers.
write_master_av_address	Out	Avalon-MM address bus of write_master port. Specifies a byte address in the Avalon-MM address space.

Table A–45. Frame Buffer Signals (Part 3 of 3)

Signal	Direction	Description
write_master_av_burstcount	Out	Avalon-MM burstcount signal of write_master port. Specifies the number of transfers in each burst.
write_master_av_clock	In	The clock signal for the write_master port. The interface operates on the rising edge of the clock signal. *(Note 1)*
write_master_av_reset	In	The reset signal for the write_master port. The interface is asynchronously reset when reset is asserted high and must be de-asserted synchronously with respect to the rising edge of the clock signal. *(Note 1)*
write_master_av_waitrequest	In	Avalon-MM waitrequest signal of write_master port. Asserted by the system interconnect fabric to cause the master port to wait.
write_master_av_write	Out	Avalon-MM write signal of write_master port. Asserted to indicate write requests from the master to the system interconnect fabric.
write_master_av_writedata	Out	Avalon-MM writedata bus of write_master port. These output lines carry data for write transfers.
writer_control_av_chipselect	In	Avalon-MM chipselect signal of writer_control slave port. The writer_control port ignores all other signals unless this signal is asserted.
writer_control_av_readdata	Out	Avalon-MM readdata bus of writer_control slave port. These output lines are used for read transfers.
writer_control_av_write	In	Avalon-MM write signal of writer_control slave port. When this signal is asserted, writer_control accepts new data from the writedata bus.
writer_control_av_writedata	In	Avalon-MM writedata bus of writer_control slave port. These input lines are used for write transfers.

Note to Table A–45:

(1) Additional clock and reset signals are available when **Use separate clocks for the Avalon-MM master interfaces** is on in the MegaWizard interface.

Line Buffer Compiler

Table A–46 shows the input and output signals for the Line Buffer Compiler MegaCore function.

Table A–46. Line Buffer Compiler Signals

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
din	In	Data input bus. Pixel data is transferred into the MegaCore function over this bus.
dout	Out	Data output bus. Pixel data is transferred out of the MegaCore function over this bus.
enable	In	Data enable. Data is latched from the input bus din and shifted through the Line Buffer Compiler when enable is high.

Clocked Video Input

Table A–47 shows the input and output signals for the Clocked Video Input MegaCore function.

Table A–47. Clocked Video Input Signals (Part 1 of 2)

Signal	Direction	Description
rst	In	The MegaCore function is asynchronously reset when rst is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the is_clk signal.
vid_clk	In	Clocked video clock. All the video input signals are synchronous to this clock.
vid_data	In	Clocked video data bus. Video data is transferred into the MegaCore function over this bus.
vid_datavalid	In	Clocked video data valid signal. This signal is asserted when a valid sample of video data is present on vid_data.
vid_locked	In	Clocked video locked signal. This signal is asserted when a stable video stream is present on the input. This signal is de-asserted when the video stream is removed.
vid_f	In	(Separate Sync Mode Only.) Clocked video field signal. For interlaced input, this signal distinguishes between field 0 and field 1. For progressive video, this signal should be deasserted.
vid_hd_sdn	In	Clocked video color plane format selection signal (in run-time switching of color plane transmission formats mode only). This signal distinguishes between sequential (when low) and parallel (when high) color plane formats.
vid_v_sync	In	(Separate Sync Mode Only.) Clocked video vertical sync signal. This signal is asserted during the vertical sync period of the video stream.
vid_h_sync	In	(Separate Sync Mode Only.) Clocked video horizontal sync signal. This signal is asserted during the horizontal sync period of the video stream.
is_clk	In	Avalon-ST clock signal for port dout and control. The MegaCore function operates on the rising edge of the is_clk signal.
is_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
is_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.
is_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
is_sop	Out	Avalon-ST startofpacket signal for port dout. This signal is asserted when the MegaCore function is starting a new frame.
is_eop	Out	Avalon-ST endofpacket signal for port dout. This signal is asserted when the MegaCore function is ending a frame.
av_address	In	Avalon-MM address bus of slave port control. Specifies a word offset into the slave address space. *(Note 1)*
av_read	In	Avalon-MM read signal of slave control port. When this signal is asserted, the control port drives new data onto the read data bus. *(Note 1)*
av_readdata	Out	Avalon-MM read data bus of slave control port. These output lines are used for read transfers. *(Note 1)*
av_write	In	Avalon-MM write signal of slave control port. When this signal is asserted, the control port accepts new data from the write data bus. *(Note 1)*

Table A–47. Clocked Video Input Signals (Part 2 of 2)

Signal	Direction	Description
av_writedata	In	Avalon-MM write data bus of slave control port. These input lines are used for write transfers. *(Note 1)*
status_update_int	Out	Avalon-MM interrupt signal of slave control port. When asserted the status registers of the MegaCore function have been updated and the master should read them to determine what has occurred. *(Note 1)*
overflow	Out	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. *(Note 1)*

Note to Table A–47

(1) These ports are present only if **Use control port** is on in the MegaWizard interface.

Clocked Video Output

Table A–48 shows the input and output signals for the Clocked Video Output MegaCore function.

Table A–48. Clocked Video Output Signals (Part 1 of 2)

Signal	Direction	Description
rst	In	The MegaCore function is asynchronously reset when rst is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the is_clk signal.
vid_clk	In	Clocked video clock. All the video input signals are synchronous to this clock.
vid_data	Out	Clocked video data bus. Video data is transferred into the MegaCore function over this bus.
vid_datavalid	Out	Clocked video data valid signal. This signal is asserted when an active picture sample of video data is present on vid_data.
vid_mode_change	Out	Clocked video mode change signal. This signal is asserted on the cycle before a mode change occurs.
vid_mode_match	Out	Clocked video mode match signal. A signal corresponding to the VidModeMatch register synchronized to vid_clk.
vid_v	Out	(Separate Sync Mode Only.) Clocked video vertical blanking signal. This signal is asserted during the vertical blanking period of the video stream.
vid_h	Out	(Separate Sync Mode Only.) Clocked video horizontal blanking signal. This signal is asserted during the horizontal blanking period of the video stream.
vid_f	Out	(Separate Sync Mode Only.) Clocked video field signal. For interlaced input, this signal distinguishes between field 0 and field 1. For progressive video, this signal is unused.
vid_v_sync	Out	(Separate Sync Mode Only.) Clocked video vertical sync signal. This signal is asserted during the vertical sync period of the video stream.
vid_h_sync	Out	(Separate Sync Mode Only.) Clocked video horizontal sync signal. This signal is asserted during the horizontal sync period of the video stream.
vid_ln	Out	(Embedded Sync Mode Only.) Clocked video line number signal. Used with the SDI MegaCore function to indicate the current line number when the vid_trs signal is asserted.
vid_trs	Out	(Embedded Sync Mode Only.) Clocked video time reference signal (TRS) signal. Used with the SDI MegaCore function to indicate a TRS, when asserted.

Table A–48. Clocked Video Output Signals (Part 2 of 2)

Signal	Direction	Description
is_clk	In	Avalon-ST clock signal for port dout and control. The MegaCore function operates on the rising edge of the is_clk signal.
is_ready	Out	Avalon-ST ready signal for port dout. This signal is asserted when the MegaCore function is able to receive data.
is_valid	In	Avalon-ST valid signal for port dout. This signal is asserted when the downstream device outputs data.
is_data	In	Avalon-ST data bus for port dout. Pixel data is transferred into the MegaCore function over this bus.
is_sop	In	Avalon-ST startofpacket signal for port dout. This signal is asserted when the downstream device is starting a new frame.
is_eop	In	Avalon-ST endofpacket signal for port dout. This signal is asserted when the downstream device is ending a frame.
av_address	In	Avalon-MM address bus of slave control port. Specifies a word offset into the slave address space. *(Note 1)*
av_read	In	Avalon-MM read signal of slave control port. When this signal is asserted, the control port drives new data onto the read data bus. *(Note 1)*
av_readdata	Out	Avalon-MM readdata bus of slave control port. These output lines are used for read transfers. *(Note 1)*
av_write	In	Avalon-MM write signal of slave control port. When this signal is asserted, the control port accepts new data from the write data bus. *(Note 1)*
av_writedata	In	Avalon-MM writedata bus of slave control port. These input lines are used for write transfers. *(Note 1)*
av_waitrequest	Out	Avalon-MM waitrequest bus of slave control port. When this signal is asserted, the control port cannot accept new transactions. *(Note 1)*
status_update_int	Out	Avalon-MM interrupt signal of slave control port. When asserted the status registers of the MegaCore function have been updated and the master should read them to determine what has occurred. *(Note 1)*
underflow	Out	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 1)*

Note to Table A–48

(1) These ports are present only if **Use control port** is on in the MegaWizard interface.

Color Plane Sequencer

Table A–49 shows the input and output signals for the Color Plane Sequencer MegaCore function.

Table A–49. Color Plane Sequencer Signals (Part 1 of 2)

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.

Table A–49. Color Plane Sequencer Signals (Part 2 of 2)

Signal	Direction	Description
dinN_data	In	Avalon-ST data bus for port dinN. Pixel data is transferred into the MegaCore function over this bus.
dinN_endofpacket	In	Avalon-ST endofpacket signal for port dinN. This signal marks the end of an Avalon-ST packet.
dinN_ready	Out	Avalon-ST ready signal for port dinN. This signal indicates when the MegaCore function is ready to receive data.
dinN_startofpacket	In	Avalon-ST startofpacket signal for port dinN. This signal marks the start of an Avalon-ST packet.
dinN_valid	In	Avalon-ST valid signal for port dinN. This signal identifies the cycles when the port should input data.
doutN_data	Out	Avalon-ST data bus for port doutN. Pixel data is transferred out of the MegaCore function over this bus.
doutN_endofpacket	Out	Avalon-ST endofpacket signal for port doutN. This signal marks the end of an Avalon-ST packet.
doutN_ready	In	Avalon-ST ready signal for port doutN. This signal is asserted by the downstream device when it is able to receive data.
doutN_startofpacket	Out	Avalon-ST startofpacket signal for port doutN. This signal marks the start of an Avalon-ST packet.
doutN_valid	Out	Avalon-ST valid signal for port doutN. This signal is asserted when the MegaCore function outputs data.

Test Pattern Generator

Table A–50 shows the input and output signals for the Test Pattern Generator MegaCore function.

Table A–50. Test Pattern Generator Signals (Part 1 of 2)

Signal	Direction	Description
clock	In	The main system clock. The MegaCore function operates on the rising edge of the clock signal.
reset	In	The MegaCore function is asynchronously reset when reset is asserted high. The reset must be de-asserted synchronously with respect to the rising edge of the clock signal.
dout_data	Out	Avalon-ST data bus for port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_endofpacket	Out	Avalon-ST endofpacket signal for port dout. This signal marks the end of an Avalon-ST packet.
dout_ready	In	Avalon-ST ready signal for port dout. This signal is asserted by the downstream device when it is able to receive data.
dout_startofpacket	Out	Avalon-ST startofpacket signal for port dout. This signal marks the start of an Avalon-ST packet.
dout_valid	Out	Avalon-ST valid signal for port dout. This signal is asserted when the MegaCore function outputs data.
control_av_address	In	Avalon-MM address bus of slave port control. Specifies a word offset into the slave address space. *(Note 1)*

Table A–50. Test Pattern Generator Signals (Part 2 of 2)

Signal	Direction	Description
control_av_chipselect	In	Avalon-MM chipselect signal of slave port control. The control port ignores all other signals unless this signal is asserted. *(Note 1)*
control_av_readdata	Out	Avalon-MM readdata bus of slave port control. These output lines are used for read transfers. *(Note 1)*
control_av_write	In	Avalon-MM write signal of slave port control. When this signal is asserted, the control port accepts new data from the writedata bus. *(Note 1)*
control_av_writedata	In	Avalon-MM writedata bus of slave port control. These input lines are used for write transfers. *(Note 1)*

Note to Table A–50

(1) These ports are present only if **Runtime control of image size** is on in the MegaWizard interface.

References

1. *International Telecommunications Union, Geneva. Recommendation ITU-R BT.601, Encoding Parameters of Digital Television for Studios, 1992.*
2. *Ken Turkowski. Graphics gems, chapter Filters for common resampling tasks, pages 147–165. Academic Press Professional, Inc., San Diego, CA, USA, 1990.*
3. *E Catmull and R Rom. A class of local interpolating splines. Computer Aided Geometric Design, pages 317–326, 1974.*
4. *MegaCore IP Library Release Notes and Errata*.
5. *AN 320: OpenCore Plus Evaluation of Megafunctions.*
6. *AN427: Video and Image Processing Up Conversion Example Design*.
7. *Quartus II Installation & Licensing for Windows.*
8. *Quartus II Installation & Licensing for Linux*.
9. *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook*.
10. *Avalon Interface Specifications*.

Revision History

The following table shows the revision history for this user guide.

Date	Version	Changes Made
March 2009	8.0	Updated for Quartus II 9.0: ■ The Deinterlacer MegaCore function supports controlled frame dropping or repeating to keep the input and output frame rates locked together ■ The Test Pattern Generator MegaCore function can generate a user-specified constant color that can be used as a uniform background ■ Preliminary support for Arria ® II GX devices
November 2008	7.0	Updated for Quartus II 8.1: ■ Added new Test Pattern Generator MegaCore function ■ The Deinterlacer MegaCore function supports pass-through mode and run-time algorithm switching ■ The Deinterlacer and Frame Buffer MegaCore functions support clock crossing for improved external memory access efficiency ■ The Clocked Video Input and Clocked Video Output MegaCore functions support run-time switching between standard definition (SD) and high definition (HD) video streams ■ The Color Space Converter MegaCore function supports run-time changing of coefficients ■ The Gamma Corrector MegaCore function supports parallel data processing for three channels ■ The 2D FIR Filter MegaCore function supports run-time changing of coefficients ■ Full support for Stratix ® III devices
July 2008	6.2	Updated for Quartus II 8.0 SP1: ■ Added control packet transfer example ■ Updated timing constraints for the Clocked Video Input and Clocked Video Output ■ Updated register map for the Clocked Video Output
June 2008	6.1	Updated on website only: ■ Added performance data ■ Added control register map for the Clipper ■ Updated the Alpha Blending Mixer functional description
May 2008	6.0	Updated for Quartus II 8.0: ■ Added new Clipper, Clocked Video Input, Clocked Video Output, Frame Buffer, and Color Plane Sequencer MegaCore functions ■ All MegaCore functions now support 2,600 pixels height and width. ■ Alpha Blending Mixer and Deinterlacer support parallel processing for 1080p60 ■ Enhanced Avalon-ST Video protocol description ■ Full support for Cyclone® III devices ■ Preliminary support for Stratix® IV devices

Date	Version	Changes Made
October 2007	5.0	Updated for Quartus II 7.2: ■ Separate design flows and parameterization chapters ■ Updated parameters and functional description for enhancements to the Color Space Converter, Chroma Resampler, and Deinterlacer
May 2007	4.0	Updated for Quartus II 7.1: ■ Updated parameters and functional description for enhancements to the Color Space Converter and Scaler ■ Updated parameters and functional description for arbitrary assignment of bit depths and resolutions enhancement to all MegaCore functions ■ Updated Interfaces chapter for enhancements to the run-time control facilities ■ Support for Arria™ GX devices
December 2006	3.0	Updated for Quartus II 7.0: ■ Preliminary support for Cyclone III devices
December 2006	2.0	Updated for Quartus II 6.1: ■ Preliminary support for Stratix III devices ■ Added new Interfaces chapter ■ Updated MegaWizard® Plug-In Manager interface ■ Updated functional description for the Scaler MegaCore® function ■ Updated functional description for the 2D FIR MegaCore Function ■ Replaced walkthrough with new tutorial procedures
April 2006	1.0	First revision of this user guide for Suite version 1.0 (Quartus II 6.0).

How to Contact Altera

For the most up-to-date information about Altera® products, refer to the following table.

Contact *(Note 1)*	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Altera literature services	Email	literature@altera.com
Non-technical support (General)	Email	nacomp@altera.com
(Software Licensing)	Email	authorization@altera.com

Note:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions that this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicates command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, and software utility names. For example, **\qdesigns** directory, **d:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicates document titles. For example, *AN 519: Stratix IV Design Guidelines.*
Italic type	Indicates variables. For example, *n* + 1. Variable names are enclosed in angle brackets (< >). For example, ***<file name>*** and ***<project name>*.pof** file.
Initial Capital Letters	Indicates keyboard keys and menu names. For example, Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. Active-low signals are denoted by suffix n. Example: `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
1., 2., 3., and a., b., c., and so on.	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The angled arrow instructs you to press the Enter key.
	The feet direct you to more information about a particular topic.