Video and Image Processing Suite

User Guide

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

Suite Version: 7.1
Document Date May 2007

UG-VIPSUITE-4.0

Contents

Chapter 3. Interfaces

Chapter 4. Specifications

About This User Guide

Revision History

The following table displays the revision history for this User Guide.

Date	Version	Changes Made
May 2007	7.1	• Updated tutorial and functional description for enhancements to the Scaler • Updated tutorial for enhancements to the Color Space Converter • Updated tutorial 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 • Added Arria™ GX support
December 2006	7.0	Added Cyclone® III support.
December 2006	6.1	• Added 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

How to Contact Altera

For the most up-to-date information about Altera® products, refer to the following table.

Information Type	Contact *Note (1)*
Technical support	www.altera.com/mysupport/
Product literature	www.altera.com
Altera literature services	literature@altera.com
FTP site	ftp.altera.com

Note to table:
(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown in the following table.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: "Typographic Conventions."
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input.` Active-low signals are denoted by suffix `n`, e.g., `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ● •	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
	The caution calls attention to a condition that could damage the product or design and should be read prior to starting or continuing with the procedure or process.
	The warning calls attention to a condition that could cause injury to the user and should be read prior to starting or continuing the procedure or processes.
↵	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.

1. About This MegaCore Function Suite

Release Information

The Altera® Video and Image Processing Suite includes the following MegaCore® functions:

- Color Space Converter
- Chroma Resampler
- Gamma Corrector
- 2D FIR Filter
- 2D Median Filter
- Alpha Blending Mixer
- Scaler
- Deinterlacer
- Line Buffer Compiler

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

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

Item	Description
Version	7.1 (All MegaCore functions)
Release Date	May 2007
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) 00B7 (Scaler) 00B8 (Line Buffer Compiler)
Vendor ID(s)	6AF7

Device Family Support

MegaCore functions provide either full or preliminary support for target Altera device families, as described below:

- *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™ GX	Preliminary
Cyclone® II	Full
Cyclone III	Preliminary
HardCopy® II	Full
Stratix®	Full
Stratix II	Full
Stratix II GX	Full
Stratix III	Preliminary
Stratix GX	Full
Other device families	No support

Features

- All the MegaCore functions in the Video and Image Processing Suite can be connected using a common Avalon® Streaming interface and image streaming protocol.
- Avalon Memory-Mapped interfaces are used 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.
- DSP Builder ready.

General Description

The Altera Video and Image Processing Suite is a collection of MegaCore functions that can be used to facilitate the development of customer 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 color space converter 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.

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. Using this fact, video transmitted in the Y'CbCr color space often subsamples the color components (Cb and Cr) to save on data bandwidth. The specification of how this subsampling is done provides a sampling format. These sampling formats 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, the Gamma Corrector MegaCore function can be programmed with a look-up-table that models the non-linear function which it then uses to transform the video data and get 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.

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 can be used to remove 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 eight image layers. The function supports both picture-in-picture mixing and image blending.

Scaler

The Scaler MegaCore function provides a means to resize and/or clip video streams. The MegaCore function supports nearest neighbor, bilinear, bicubic, and polyphase scaling algorithms. It can be configured to change resolutions and/or filter coefficients at runtime using an Avalon Memory-Mapped (Avalon-MM) Slave interface.

Deinterlacer

The Deinterlacer MegaCore function converts interlaced video to progressive video using either "Bob" or "Weave" algorithms. Interlaced video is commonly used in television standards such as phase alternation line (PAL) and national television system committee (NTSC), but progressive video is required by LCD displays and is often more useful for subsequent image processing functions.

Line Buffer Compiler

The Line Buffer Compiler MegaCore function can be used to efficiently map video line buffers to Altera on-chip memories.

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

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*.

DSP Builder Support

Altera's DSP Builder product shortens digital signal processing (DSP) design cycles by helping you create the hardware representation of a DSP design in an algorithm-friendly development environment.

DSP Builder integrates the algorithm development, simulation, and verification capabilities of The MathWorks MATLAB® and Simulink® system-level design tools with Altera Quartus® II software and third-party synthesis and simulation tools. You can combine existing Simulink blocks with Altera DSP Builder MegaCore function variation blocks to verify system level specifications and perform simulation.

After installing any of the Video and Image Processing Suite MegaCore functions, a Simulink symbol for the MegaCore function appears in the Simulink library browser. The MegaCore functions are available from the Altera DSP Builder blockset in the **Video and Image Processing** library.

The Line Buffer Compiler MegaCore function is not supported in DSP Builder.

DSP Builder also supports integration with SOPC Builder using Avalon® Memory-Mapped (Avalon-MM) master/slave and Avalon Streaming (Avalon-ST) source/sink interfaces. The Video and Image Processing Suite MegaCore functions have built-in Avalon-MM and Avalon-ST interfaces that allow you to manually insert hardware boundary blocks to define the contents of an SOPC Builder **class.ptf** file.

For information on DSP Builder, refer to the *DSP Builder User Guide* and *DSP Builder Reference Manual*. For information about SOPC Builder, refer to volume 4 of the *Quartus II Handbook*. Refer to the *Avalon Streaming Interface Specification* and the *Avalon Memory-Mapped Interface Specification* for more information about these interface types.

OpenCore Plus Evaluation

With Altera's free OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP^{SM} megafunction) within your system
- Verify the functionality of your design, as well as quickly and easily 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 only need to purchase a license for the MegaCore function when you are completely satisfied with its functionality and performance, and want to take your design to production.

For more information on OpenCore Plus hardware evaluation using MegaCore functions, see "OpenCore Plus Time-Out Behavior" on page 4–30 and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Performance

This section shows typical expected performance for the Video and Image Processing Suite MegaCore functions when using the Quartus II software version 7.1.

Cyclone III devices use logic elements; 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	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks (9x9)	DSP Blocks (18x18)	Memory M9K	f_{MAX} (MHz)
Converting 1024×768 14-bit Y'UV to Computer R'G'B' using 18-bit coefficients and 15-bit summands. 14_1024x768_18c_YUV2cRGB								
	Cyclone III	662	—	—	6	—	—	193
	Stratix III	—	296	535	—	6	—	304
Converting 1,080 pixel 10-bit Studio R'G'B' to HDTV Y'CbCr using 18-bit coefficients and 27-bit summands. 10b_1920x1080_sRGB2HDYcc_18b								
	Cyclone III	559	—	—	6	—	—	193
	Stratix III	—	316	510	—	6	—	314
Converting 720×576 8-bit Computer R'G'B' to Y'UV using 9-bit coefficients and 8-bit summands. 8bit_720x576_cRGBtoYUV_9bit								
	Cyclone III	408	—	—	3	—	—	193
	Stratix III	—	216	353	—	3	—	322
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. csc_8b_640x480_SDYcc2cRGB_9bit_p								
	Cyclone III	758	—	—	9	—	—	243
	Stratix III	—	351	536	—	9	—	383

Chroma Resampler

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

Table 1–4. Chroma Resampler Performance

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks (9x9)	DSP Blocks (18x18)	Memory M9K	f_{MAX} (MHz)
Downsampling to 4:2:2 using linear interpolation, working on 64×64 frames with 8-bit data.								
	Cyclone III	163	—	—	—	—	—	293
	Stratix III	—	79	140	—	—	—	517
Downsampling to 4:2:0 using linear horizontal interpolation, and no vertical interpolation. Video frames are 256×256 with 10-bit data.								
	Cyclone III	299	—	—	—	—	2	257
	Stratix III	—	154	211	—	—	2	398
Upsampling from 4:2:2 using linear interpolation, working on 1,080 pixel frames with 12-bit data.								
	Cyclone III	379	—	—	—	—	—	241
	Stratix III	—	194	309	—	—	—	400
Upsampling from 4:2:0 using no interpolation. Video frames are 352×288 with 10-bit data.								
	Cyclone III	304	—	—	—	—	3	253
	Stratix III	—	176	194	—	—	3	390

Gamma Corrector

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

Table 1–5. Gamma Corrector Performance (Part 1 of 2)

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks (9x9)	DSP Blocks (18x18)	Memory M9K	f_{MAX} (MHz)
Gamma correcting 64×64 three color 8-bit data.								
	Cyclone III	132	—	—	—	—	1	259
	Stratix III	—	79	98	—	—	1	496
Gamma correcting 720×576 one color 10-bit data.								
	Cyclone III	152	—	—	—	—	3	259
	Stratix III	—	91	116	—	—	3	490

Table 1–5. Gamma Corrector Performance (Part 2 of 2)

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks (9x9)	DSP Blocks (18x18)	Memory M9K	f_{MAX} (MHz)
Gamma correcting 128×128 three color 8-bit data.								
	Cyclone III	134	—	—	—	—	1	259
	Stratix III	—	82	101	—	—	1	517
Gamma correcting 1,080 pixel one color 10-bit data.								
	Cyclone III	154	—	—	—	—	3	259
	Stratix III	—	91	117	—	—	3	472

2D FIR Filter

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

Table 1–6. 2D FIR Filter Performance

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks (9x9)	DSP Blocks (18x18)	Memory M9K	f_{MAX} (MHz)
Smoothing 3×3 symmetric filter, working on 640×480 8-bit R'G'B', using 9 bit coefficients.								
	Cyclone III	985	—	—	6	—	4	230
	Stratix III	—	613	691	—	4	4	351
Edge detecting 3×3 asymmetric filter, working on 352×288 8-bit R'G'B', using 3 bit coefficients								
	Cyclone III	1,036	—	—	9	—	4	224
	Stratix III	—	601	630	—	9	4	354
Sharpening 5×5 symmetric filter, working on 640×480 in 8-bit R'G'B', using 9 bit coefficients.								
	Cyclone III	1,824	—	—	12	—	8	210
	Stratix III	—	1,037	1,249	—	8	8	336
Smoothing 7×7 symmetric filter, working on 1,280×720 in 10-bit R'G'B', using 15 bit coefficients								
	Cyclone III	3,712	—	—	20	—	30	182
	Stratix III	—	1,999	2,704	—	20	30	317

2D Median Filter

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

Table 1–7. 2D Median Filter Performance

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks		Memory M9K	f_{MAX} (MHz)
					(9x9)	(18x18)		
Median filtering 64×64 pixel R'G'B frames using a 3×3 kernel of pixels.								
	Cyclone III	1,520	—	—	—	—	2	228
	Stratix III	—	804	1,056	—	—	2	343
3×3 median filtering HDTV 720 pixel monochrome video.								
	Cyclone III	1,613	—	—	—	—	6	216
	Stratix III	—	867	1,132	—	—	6	319
Median filtering 352×288 pixel two color frames using a 5×5 kernel of pixels.								
	Cyclone III	5,442	—	—	—	—	8	186
	Stratix III	—	2,486	3,699	—	—	8	276
7×7 median filtering 352×288 pixel monochrome video.								
	Cyclone III	10,979	—	—	—	—	6	175
	Stratix III	—	4,741	7,200	—	—	6	255

Alpha Blending Mixer

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

Table 1–8. Alpha Blending Mixer Performance (Part 1 of 2)

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks		Memory M9K	f_{MAX} (MHz)
					(9x9)	(18x18)		
Drawing a 64×64 pixel picture-in-picture window over the top of a 128×128 pixel background image in 8-bit R'G'B' color.								
	Cyclone III	380	—	—	—	—	1	214
	Stratix III	—	237	287	—	—	1	346
Rendering two 64×64 pixel images over 352×240 pixel background 8-bit R'G'B' video.								
	Cyclone III	579	—	—	—	—	1	224
	Stratix III	—	348	448	—	—	1	346

Table 1–8. Alpha Blending Mixer Performance (Part 2 of 2)

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks (9x9)	DSP Blocks (18x18)	Memory M9K	f_{MAX} (MHz)
Alpha blending an on-screen display within a 176×120 pixel 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	637	—	—	4	—	1	210
	Stratix III	—	327	450	—	4	1	324
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	1,640	—	—	12	—	1	182
	Stratix III	—	669	1,246	—	12	1	284

Scaler

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

Table 1–9. Scaler Performance

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks (9x9)	DSP Blocks (18x18)	Memory M9K	f_{MAX} (MHz)
Scaling 640×480, 8-bit, three color data up to 1,024×768 with linear interpolation. This could be used to convert VGA to VESA 1024×768.								
	Cyclone III	4,061	—	—	4	—	6	203
	Stratix III	—	422	470	—	4	6	366
Scaling RGB Quarter common intermediate format (QCIF) to common intermediate format (CIF) with no interpolation.								
	Cyclone III	390	—	—	—	—	3	235
	Stratix III	—	186	190	—	—	3	414
Scaling NTSC standard definition (720x480) RGB to high definition 1080p using a bicubic algorithm.								
	Cyclone III	1,040	—	—	—	—	3	248
	Stratix III	—	792	943	—	8	12	314
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 runtime control interface.								
	Cyclone III	3,408	—	—	19	—	—	176
	Stratix III	—	1,754	2,321	—	19	62	286

Deinterlacer

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

Table 1–10. Deinterlacer Performance

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks (9x9)	DSP Blocks (18x18)	Memory M9K	f_{MAX} (MHz)
Deinterlacing 64×64 pixel 8-bit R'G'B' frames using Bob with scanline duplication.								
	Cyclone III	179	—	—	—	—	1	259
	Stratix III	—	119	145	—	—	1	475
Bob deinterlacing with scanline interpolation working on 352×288 pixel 12-bit Y'CbCr 4:2:2 frames.								
	Cyclone III	348	—	—	—	—	2	259
	Stratix III	—	254	273	—	—	2	456
Deinterlacing HDTV 1080i resolution with 12-bit Y'CbCr 4:4:4 color using the Weave algorithm.								
	Cyclone III	1,309	—	—	—	—	6	165
	Stratix III	—	982	1,003	—	—	6	247
Bob deinterlacing with scanline interpolation working on 176×144 pixel 12-bit monochrome video.								
	Cyclone III	324	—	—	—	—	1	259
	Stratix III	—	221	254	—	—	1	471

Line Buffer Compiler

Table 1–11 shows the performance figures for the Line Buffer Compiler.

Table 1–11. Line Buffer Compiler Performance (Part 1 of 2)

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks (9x9)	DSP Blocks (18x18)	Memory M9K	f_{MAX} (MHz)
Three 8-bit wide 64 pixel lines.								
	Cyclone III	23	—	—	—	—	1	259
	Stratix III	—	21	22	—	—	1	599
Two 8-bit wide 720 pixel lines. This configuration would be appropriate for 8-bit PAL line buffering.								
	Cyclone III	37	—	—	—	—	4	259
	Stratix III	—	40	30	—	—	4	599
Four 10-bit wide 1,280 pixel lines. This parameterization might be used for 10-bit HDTV 720 pixel line buffering.								
	Cyclone III	35	—	—	—	—	12	259
	Stratix III	—	37	31	—	—	12	582

Table 1–11. Line Buffer Compiler Performance (Part 2 of 2)

	Device Family	Logic Elements	Combinational ALUTs	Logic Registers	DSP Blocks		Memory M9K	f_{MAX} (MHz)
					(9x9)	(18x18)		
Two 10-bit wide 1,920 pixel lines. This configuration could be used in an HDTV 1080i system.								
	Cyclone III	33	—	—	—	—	9	259
	Stratix III	—	34	31	—	—	12	582

2. Getting Started

Design Flow

To evaluate a Video and Image Processing Suite MegaCore® function using the OpenCore Plus feature, include these steps in your design flow:

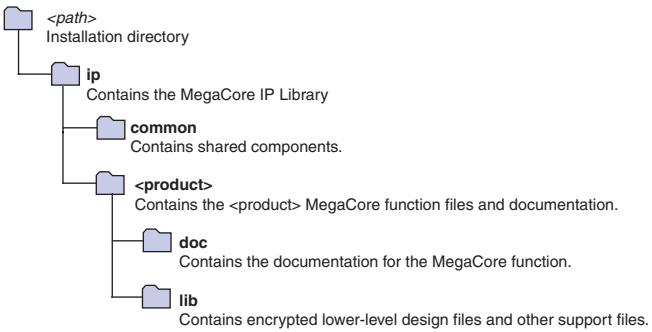
1. Obtain and install the MegaCore function.

The Video and Image Processing Suite is part of the MegaCore IP Library, which is distributed with the Quartus® II software and downloadable from the Altera website: **www.altera.com**

For system requirements and install instructions, refer to the *Quartus II Installation & Licensing for Windows* manual or the *Quartus II Installation & Licensing for UNIX and Linux* manual on the Altera Literature website.

Figure 2–1 shows the directory structure for a typical Video and Image Processing Suite MegaCore function where <path> is the installation directory. The default installation directory on Windows is **c:\altera\71**; or on UNIX and Linux, the default installation directory is **/opt/altera/71**.

Figure 2–1. Directory Structure



2. Create a custom variation of the MegaCore function.
3. Implement the rest of your design using the design entry method of your choice.
4. Use the IP functional simulation model to verify the operation of your design.

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*.

5. Use the Quartus II software to compile your design.
6. Generate an OpenCore Plus time-limited programming file, which you can use to verify the operation of your design in hardware on the Altera® DSP development board.
7. Program the Altera device or devices with the completed design.
8. Perform design verification.
9. Purchase a license for the MegaCore function.

After you have purchased a license for the MegaCore function, the design flow requires the following additional steps:

1. Set up licensing.
2. Generate a programming file for the Altera® device or devices on your board.
3. Program the Altera device or devices with the completed design.

Video and Image Processing Suite Tutorial

This tutorial explains how to create a Video and Image Processing Suite MegaCore function variation using the Altera MegaWizard® Plug-In Manager and the Quartus II software. When you have finished generating a MegaCore function variation, you can incorporate it into your overall project.

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

This tutorial involves the following steps:

- Create a New Quartus II Project
- Launch the MegaWizard Plug-In Manager
- Parameterize
- Set Up Simulation
- Generate Files

Create a New Quartus II Project

You need to create a new Quartus II project with the **New Project Wizard**, which specifies the working directory for the project, assigns the project name, and designates the name of the top-level design entity.

To create a new project, follow these steps:

1. Choose **Programs > Altera > Quartus II** <*version*> (Windows Start menu) to run the Quartus II software. Alternatively, you can use the Quartus II Web Edition software.

2. Choose **New Project Wizard** (File menu).

3. Click **Next** in the **New Project Wizard Introduction** page (this page does not display if you turned it off previously).

4. In the **New Project Wizard: Directory, Name, Top-Level Entity** page, enter the following information:

   a. Specify the working directory for your project. For example, this tutorial uses the **d:\mydesigns\vip** directory.

   b. Specify the name of the project. This tutorial uses **example** for the project name.

   The Quartus II software automatically specifies a top-level design entity that has the same name as the project. This tutorial assumes that the names are the same.

5. Click **Next** to close this page and display the **New Project Wizard: Add Files** page.

   When you specify a directory that does not already exist, a message asks you if the specified directory should be created. Click **Yes** to create the directory.

6. If you installed the MegaCore IP Library in a different directory from where you installed the Quartus II software, you must add the user library:

   a. Click **User Libraries**.

   b. Type <*path*>`\ip` in the **Library name** box, where <*path*> is the directory in which you installed the MegaCore IP library.

   c. Click **Add** to add the path to the Quartus II project.

d. Click **OK** to save the library path in the project.

7. Click **Next** to close this page and display the **New Project Wizard: Family & Device Settings** page.

8. On the **New Project Wizard: Family & Device Settings** page, choose the target device family in the **Family** list. For example, `Stratix III`.

9. The remaining pages in the **New Project Wizard** are optional. Click **Finish** to complete the Quartus II project.

You have finished creating your new Quartus II project.

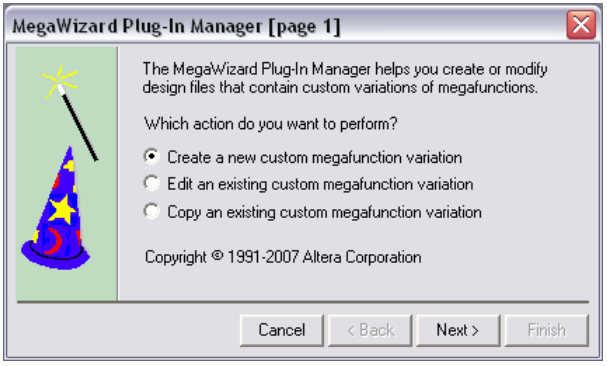
Launch the MegaWizard Plug-In Manager

To launch the MegaWizard Plug-In Manager in the Quartus II software, follow these steps:

1. Start the MegaWizard Plug-In Manager by choosing **MegaWizard Plug-In Manager** (Tools menu). The **MegaWizard Plug-In Manager** dialog box is displayed (Figure 2–2).

Refer to the Quartus II Help for more information on how to use the MegaWizard Plug-In Manager.

Figure 2–2. MegaWizard Plug-In Manager



2. Specify that you want to create a new custom megafunction variation and click **Next**.

3. Expand the **DSP** directory under **Installed Plug-Ins** by clicking the + icon next to the name.

4. Expand the **Video and Image Processing** folder and select <*function*><*version*> for the required MegaCore function in this directory.

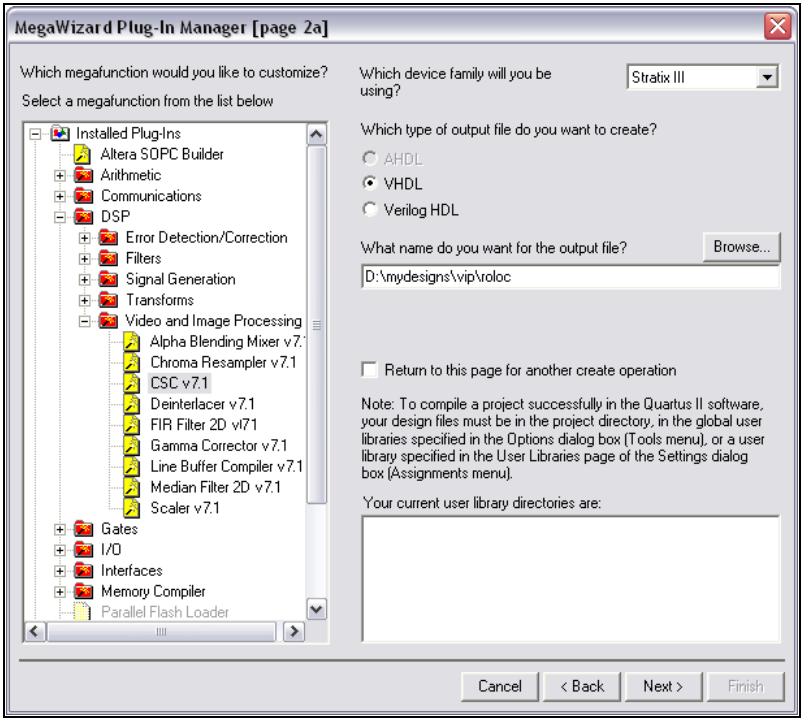
5. Check that the device family is the same as you specified in the **New Project Wizard**.

6. Choose the top-level output file type for your design; the MegaWizard interface supports VHDL, and Verilog HDL.

7. The MegaWizard Plug-In Manager shows the project path that you specified in the **New Project Wizard**. Append a variation name for the MegaCore function output files <*project path*>\<*variation name*>.

   Figure 2–3 shows the MegaWizard interface after you have made these settings.

Figure 2–3. Select the MegaCore Function



8. Click **Next** to display the **Parameter Settings** page for the selected MegaCore function.

You can change the page that is displayed by clicking **Next** or **Back**. You can move directly to a named page by clicking **Parameter Settings**, **Simulation Model**, **Summary**, or the name of an individual parameter setting page.

Parameterize

To parameterize your Video and Image Processing Suite MegaCore function follow the steps in the following sections:

- "Color Space Converter" on page 2–6
- "Chroma Resampler" on page 2–10
- "Gamma Corrector" on page 2–12
- "2D FIR Filter" on page 2–13
- "2D Median Filter" on page 2–18
- "Alpha Blending Mixer" on page 2–19
- "Scaler" on page 2–20
- "Deinterlacer" on page 2–25
- "Line Buffer Compiler" on page 2–27

The parameters available depend on the MegaCore function you have selected.

Color Space Converter

A typical application for a Color Space Converter would be to convert Y'CbCr standard definition television images to R'G'B' for display on a computer monitor. To configure the MegaCore function to perform this function, follow these steps:

1. Set the image format in the **Parameter Settings: General** page (see Figure 2–4 on page 2–7) by choosing the image width and height:

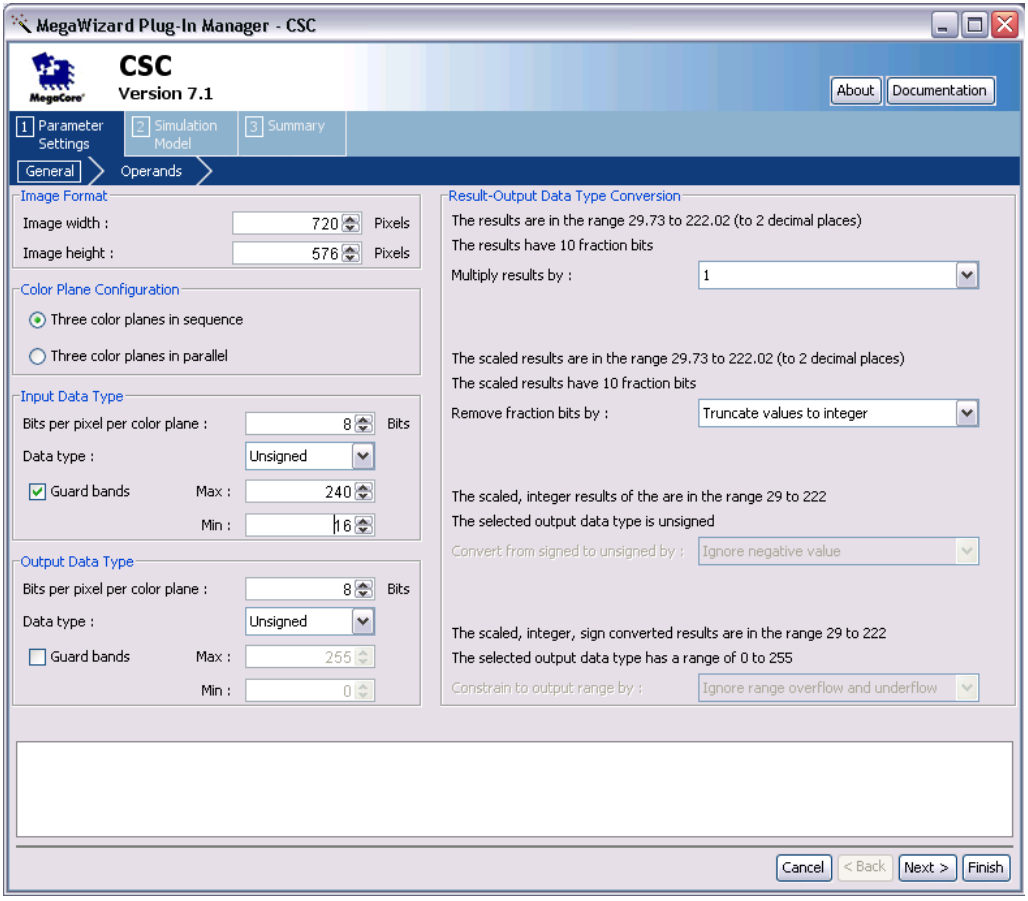
   - **Image width**: `720`
   - **Image height**: `576`

   This is the resolution of Phase Alternation Line (PAL) video, a common standard definition television format.

2. Set the color plane configuration to **Three color planes in sequence**.

   This assumes the core will be receiving data channels in sequence, not parallel.

Figure 2–4. General Parameter Settings for the Color Space Converter



3. Set the input data type:

   - **Bits per pixel per color plane**: 8 Bits
   - **Data type**: `Unsigned`
   - **Guard bands**: `On`
   - **Max**: `240`
   - **Min**: `16`

   This assumes that the core will never receive data in the guard bands, in this case between 241 to 255 and from 0 to 15.

4. Set the output data type:

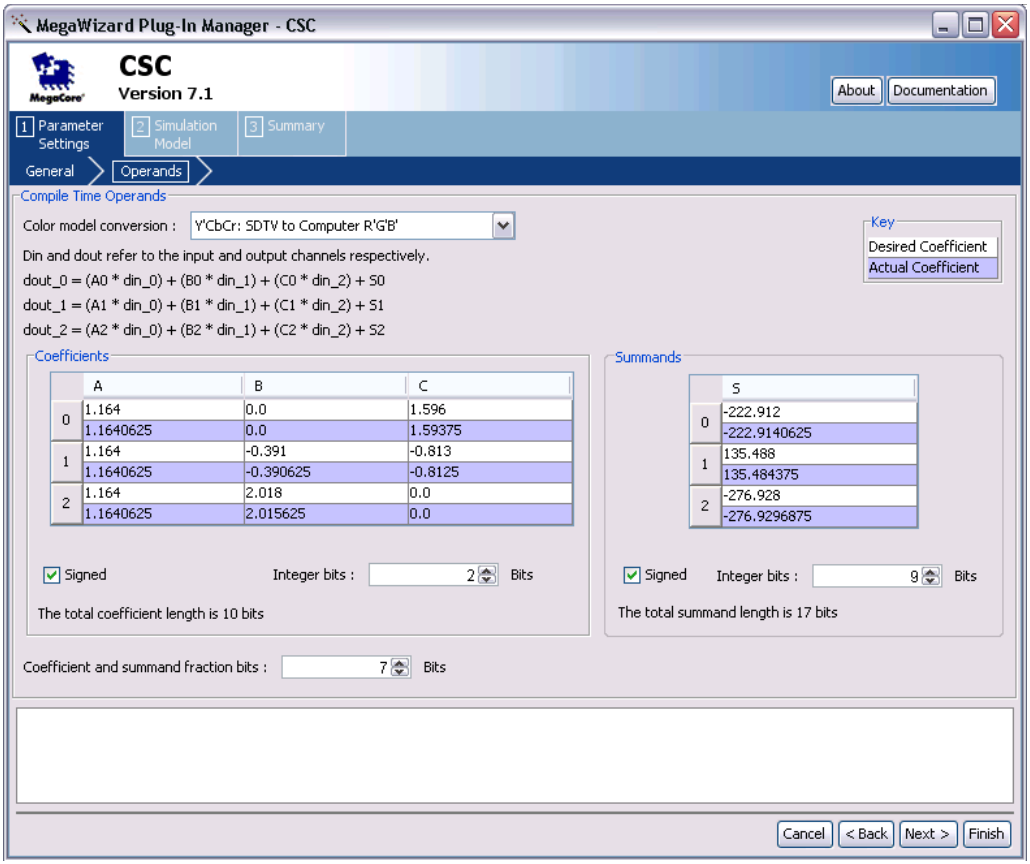
   - **Bits per pixel per color plane**: `8 Bits`
   - **Data type**: `Unsigned`
   - **Guard bands**: `Off`

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

See Table 4–10 on page 4–30 for more information about the options on the Color Space Converter **Parameter Settings: General** page.

5. Click **Next** to display the **Parameter Settings: Operands** page (Figure 2–5).

Figure 2–5. Coefficients Parameter Settings for the Color Space Converter



6. Choose `Y'CbCr: SDTV to Computer R'G'B'` from the list of predefined color model conversion options.

   Notice that the coefficient values are updated to preset values. Editing these values would cause the color model conversion option to change to `Custom`.

7. Set the precision for coefficients and summands:

   - **Coefficients Signed**: `On`
   - **Coefficients Integer bits**: `2`
   - **Summands Signed**: `On`
   - **Summands Integer bits**: `9`
   - **Coefficient and summand fraction bits**: `7`

   Notice how the actual constants (purple cells) change: Turning on signed, allows negative values; increasing integer bits, increases the magnitude range; increasing the fraction bits, increases the precision.

See Table 4–11 on page 4–32 for more information about the options on the Color Space Converter **Parameter Settings: Operands** page.

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

   Notice that after changing the coefficients, the ranges shown in the Result-Output Data Type Conversion section have changed.

9. Set the **Multiply results by** option to 1

10. Change the **Remove fraction bits by** option to `Round values to nearest integer`.

    Notice that the “scaled integer results” no longer have a fractional part.

11. Set the **Convert from signed to unsigned by** option to `Replacing negative values with zero`.

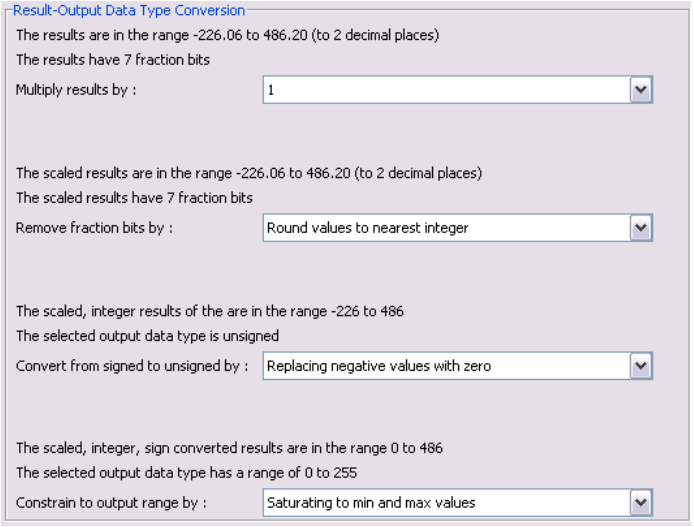
    Notice that the “scaled, integer, sign converted results" no longer include negative values.

12. Set the **Constrain to output range by** option to `Saturating to min and max values`.

    Notice that the label shows the result range prior to constraining, it also shows the output type has a range smaller than this. Therefore the results must be constrained to this range.

Figure 2–6 shows the updated Result-Output Data Type Conversion section in the **Parameter Settings: General** page.

Figure 2–6. Updated General Parameter Settings for the Color Space Converter



If a set of custom coefficient is required, these can be typed into the white cells in the tables on the **Parameter Settings**: **Operands** page.

Custom coefficients can also be pasted into the table from a spreadsheet (such as Microsoft Excel). Blank lines must be left in your input data for the non editable cells.

13. Click **Next** twice to complete the parameterization and display the **Simulation Model** page (Figure 2–21 on page 2–28).

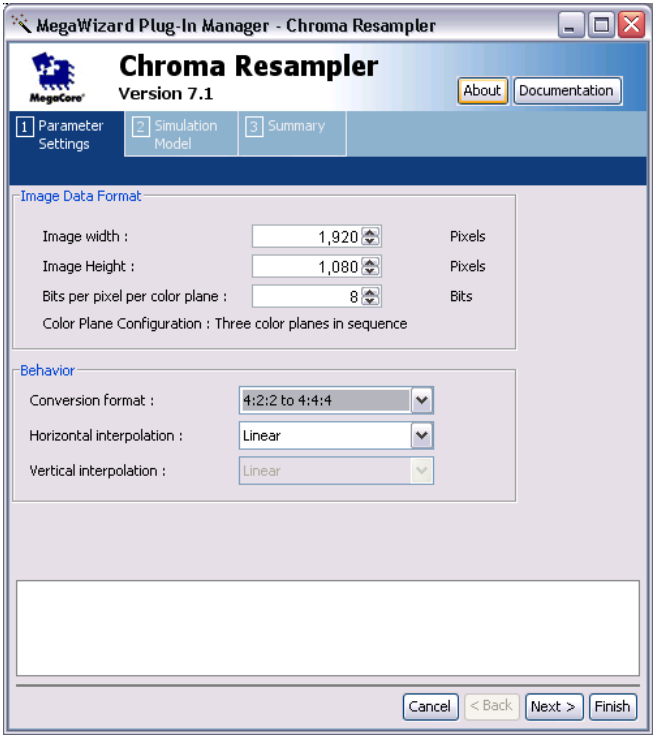
Chroma Resampler

To configure your Chroma Resampler to upsample a high-definition 4:2:2 video stream, follow these steps:

1. Set the image data format in the **Parameter Settings** page (see Figure 2–7 on page 2–11) by choosing the image resolution and the number of bits per pixel per color plane:
   - **Image width**: `1920`
   - **Image height**: `1080`
   - **Bits per pixel per color plane**: `8`

The color plane configuration is always set to `Three planes in sequence` for this MegaCore function.

Figure 2–7. Parameter Settings for the Chroma Resampler



2. Set the following conversion format and interpolation behavior:

   - **Conversion format**: `4:2:2 to 4:4:4`
   - **Horizontal interpolation**: `Linear`

   Notice that the `Vertical interpolation` control is disabled when the format conversion does not involve 4:2:0.

See Table 4–12 on page 4–33 for more information about the options on the Chroma Resampler **Parameter Settings** page.

3. Click **Next** to complete the parameterization and display the **Simulation Model** page (Figure 2–21 on page 2–28).

Gamma Corrector

To configure your Gamma Corrector to correct a 704x480 monochrome video stream for the properties of a display device, follow these steps:

1. Set the image data format in the **Parameter Settings** page (see Figure 2–8) by choosing the image resolution, the number of bits per pixel per color plane, and the number of color planes that are received and transmitted in sequence:
   - **Image width**: `704`
   - **Image height**: `480`
   - **Bits per pixel per color plane**: `8`
   - **Number of color planes in sequence**: `1`

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

See Table 4–13 on page 4–33 for more information about the options on the Gamma Corrector **Parameter Settings** page.

Figure 2–8. Parameter Settings for the Gamma Corrector

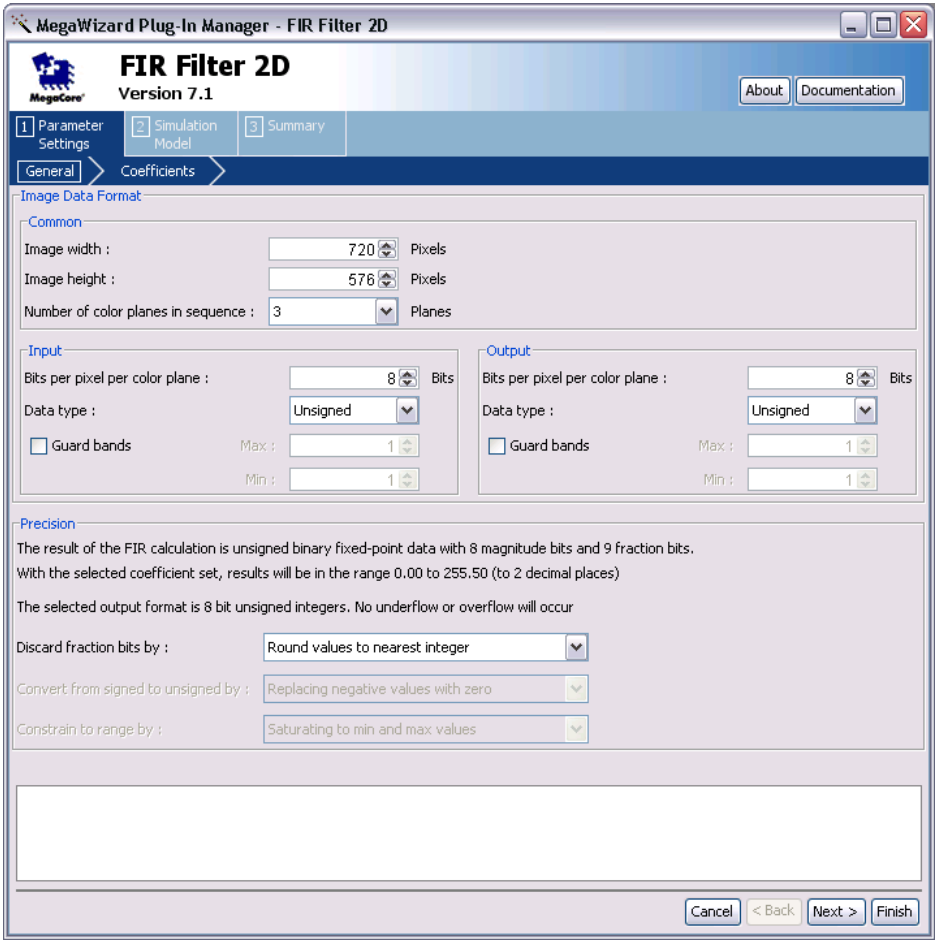
2. Click **Next** to complete the parameterization and display the **Simulation Model** page (Figure 2–21 on page 2–28).

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 operation follow these steps

1. Set the image width to `720` and the image height to `576` in the **Parameter Settings: General** page (see Figure 2–9). This is the resolution of PAL video, a common standard definition television format.

Figure 2–9. General Parameter Settings for the 2D FIR Filter



2. Set the number of color planes that are received and transmitted in sequence to be 3.

3. Set input options for the number of bits per pixel per color plane, unsigned or signed 2's complement data type, and the maximum and minimum values for the input guard bands:

   - **Bits per pixel per color plane**: 8
   - **Data type**: Unsigned
   - **Guard bands**: Off

   Turning the guard bands option off assumes that the entire 8-bit input range is used for video data. That is, the minimum value is 0 and the maximum value is 255.

4. Set output options for the number of bits per pixel per color plane, unsigned or signed 2's complement data type, and the maximum and minimum values for the output guard bands:

   - **Bits per pixel per color plane**: 8
   - **Data type**: Unsigned
   - **Guard bands**: Off

   When the guard bands option is turned off, the entire output range is allowed for video data.

   The Precision section displays three lines of information about the configuration of the 2D FIR Filter that you have specified:

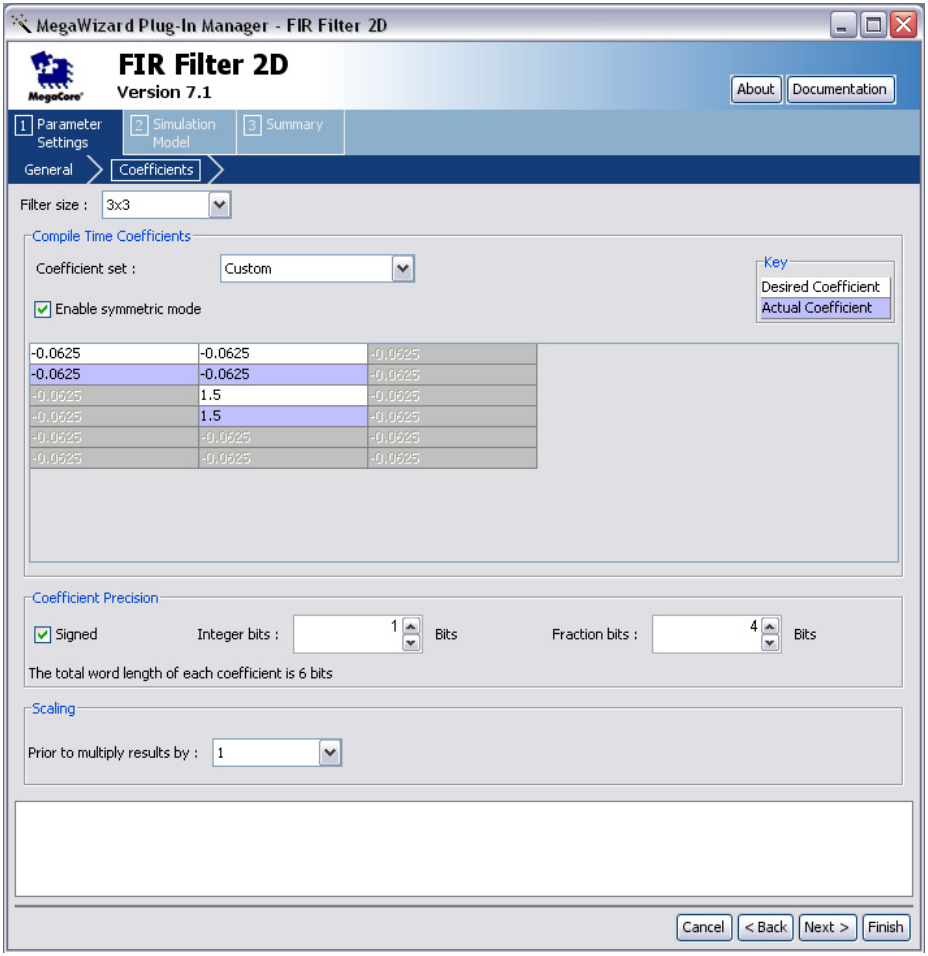
   - The first line describes the data type of the FIR calculation result before any output type conversion.
   - The second line gives the data range of the FIR calculation result before any output type conversion.
   - The third line describes the output type, and states how any potential underflow or overflow on the output of the FIR filter is handled.

5. Choose the Precision option to discard fractional bits by choosing Round values to nearest integer.

See Table 4–15 on page 4–34 for more information about the options on the 2D FIR Filter **Parameter Settings: General** page.

6. Click **Next** to display the **Parameter Settings: Coefficients** page (Figure 2–10 on page 2–15).

Figure 2–10. Coefficients Parameter Settings for the 2D FIR Filter



7. Choose a filter size of `3×3`.

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

8. Choose `Simple Smoothing` from the drop-down list of predefined coefficient sets.

   Notice that the values in the coefficient grid change when you choose 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.

9. Turn on **Enable symmetric mode**.

   Notice that when symmetric mode is set, a limited number of matrix cells are editable and many of the values are automatically inferred. For example in Figure 2–10, 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 need to be performed in the hardware.

   If you are setting custom coefficients, you can disable symmetric mode to allow edits to any of the desired coefficient values.

10. Edit the desired coefficients so that the sharpening is less strong as follows:

    - Set the central desired coefficient to: `1.5`
    - Set the top left desired coefficient to: `-0.0625`
    - Set the top central desired coefficient to: `-0.0625`

    Notice that the coefficient set changes to `Custom`.

11. Set the following options for Coefficient Precision:

    - **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 in the purple boxes should now be a close match to the desired coefficients in the white boxes.

12. Set the scaling factor **Prior to multiply results by** to `1`.

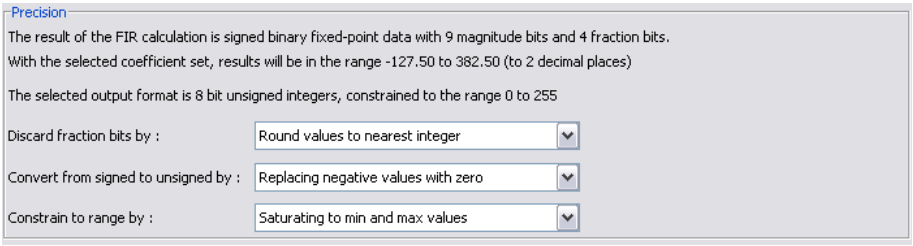
See Table 4–16 on page 4–35 for more information about the options on the 2D FIR Filter **Parameter Settings: Coefficients** page.

13. Click **Back** to re-display the **Parameter Settings: General** page.

Notice that after changing the coefficients the result range has changed and now includes negative values. This has enabled the **Convert from signed to unsigned** option.

The result range is also beyond the minimum and maximum values allowed by the output type. This has enabled the **Constrain to range** option. (See Figure 2–11.)

Figure 2–11. Updated Precision Section in the Parameter Settings: General Page



14. Set the method for converting signed to unsigned by choosing: **Replacing negative values with zero**.

15. Set the method for constraining to range by choosing: `Saturating to min and max values`.

    This will saturate the sign converted results to the minimum and maximum values allowed by the output data type (or output guard bands, when specified).

16. It may be decided at a later date that a higher precision output is required. For example, if 12 bits per pixel per color plane are required:

    a. Set the output **Bits per pixel per color plane** to `12`.

    b. Navigate to the **Parameter Settings: Coefficients** page.

    c. Set the **Prior to multiply results by** to `16`

    d. Navigate to the **Parameter Settings: General** page.

    Notice that the result range has increased. The result now has no fraction bits causing the discard fraction bits control to be disabled.

17. Click **Next** to complete the parameterization and display the **Simulation Model** page (Figure 2–21 on page 2–28).

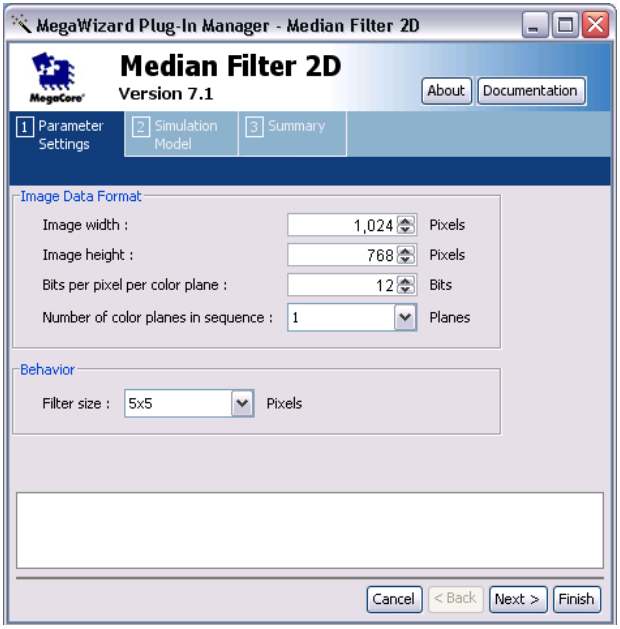
2D Median Filter

To configure your 2D Median Filter for 5×5 filtering of an example high resolution monochrome image format, follow these steps:

1. Set the image data format in the **Parameter Settings** page (see Figure 2–12) by choosing the image resolution, the number of bits per pixel per color plane, and the number of color planes that are received and transmitted in sequence:
   - **Image width**: 1024
   - **Image height**: 768
   - **Bits per pixel per color plane**: 12
   - **Number of color planes in sequence**: 1
2. Choose the filter size to 5×5.

See Table 4–17 on page 4–36 for more information about the options on the 2D Median Filter **Parameter Settings** page.

Figure 2–12. Parameter Settings for the 2D Median Filter



3. Click **Next** to complete the parameterization and display the **Simulation Model** page (Figure 2–21 on page 2–28).

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, follow these steps:

1. Use the **Parameter Settings** page to set the number of bits per pixel per color plane, the number of color planes that are received and transmitted in sequence, and the number of layers being mixed:

   - **Bits per pixel per color plane**: 8
   - **Number of color planes in sequence**: 3
   - **Number of layers being mixed**: 3
   - **Alpha blending**: On
   - **Alpha bits per pixel**: 4

   Notice that the option to specify the number of bits for alpha blending is available when you set alpha blending on. Setting this option to 4 ensures that the background layer and layer 2 are not completely obscured. The dialog box page allows three input layer resolutions (background, layer 2, and foreground) to be set and the entry fields for the remaining layers are dimmed. (See Figure 2–13 on page 2–20.)

2. Choose the required resolution for each layer:

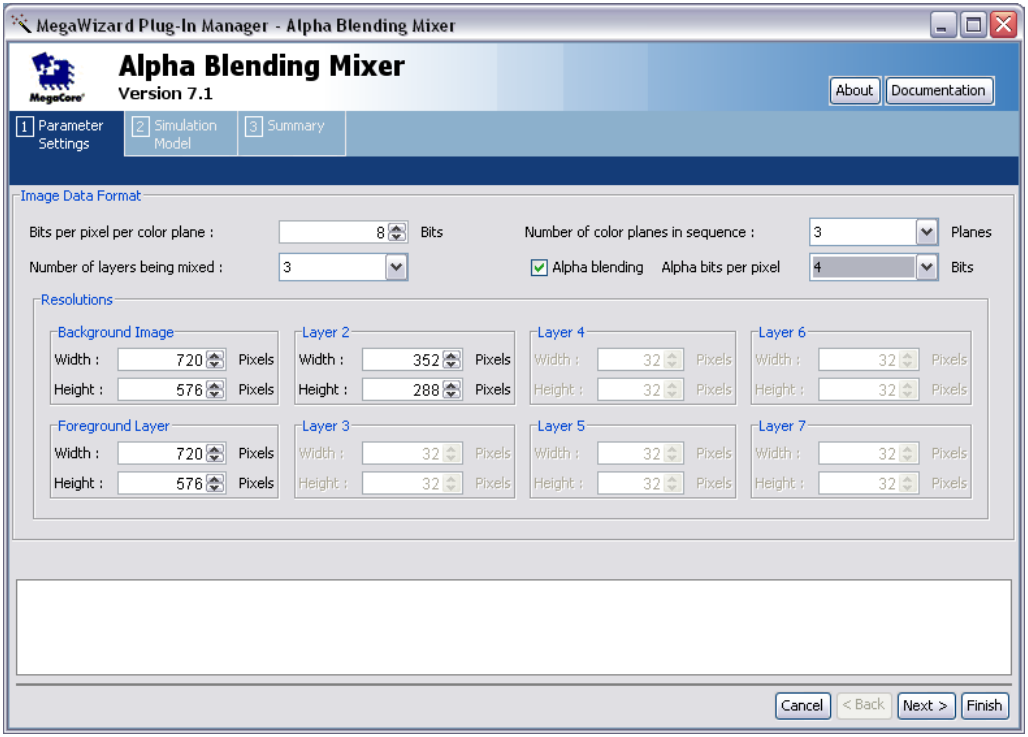
   - **Background Layer Width**: 720
   - **Background Layer Height**: 576
   - **Layer 2 Width**: 352
   - **Layer 2 Height**: 288
   - **Foreground Layer Width**: 720
   - **Foreground Layer Height**: 576

   This background resolution is the resolution of PAL video (a common standard television format). The Level 2 resolution is close to quarter screen size (suitable for a picture-in-picture window). Setting the foreground image resolution to the same value as the background screen resolution makes it easy to draw on-screen display information anywhere on the screen.

See Table 4–18 on page 4–37 for more information about the options on the Alpha Blending Mixer **Parameter Settings** page.

Figure 2–13. Parameter Settings for the Alpha Blending Mixer



3. Click **Next** to complete the parameterization and display the **Simulation Model** page (Figure 2–21 on page 2–28).

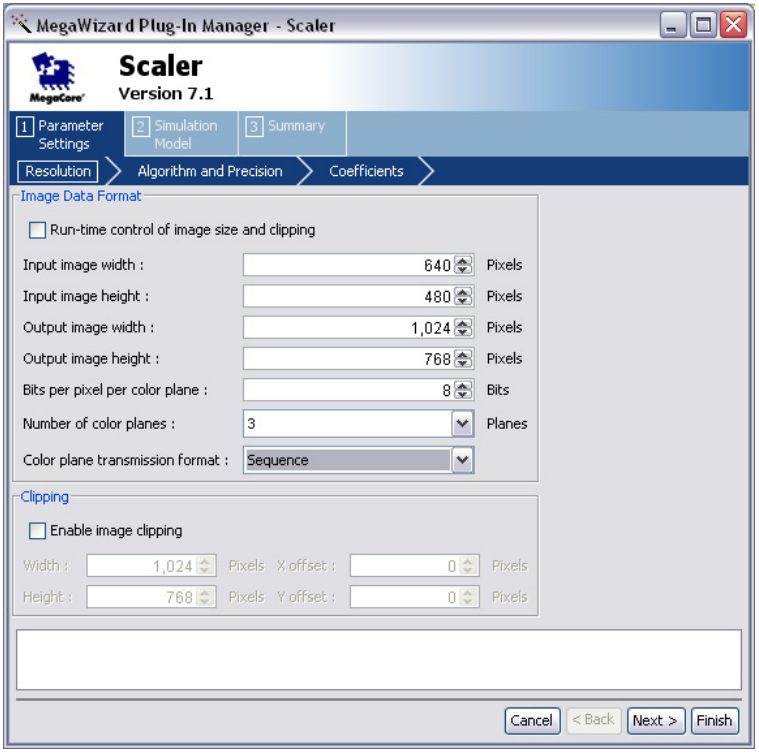
Scaler

The Scaler MegaCore function can be used to resize video streams and apply brightness coefficients.

For example, to resize a 640×480 resolution video stream to a resolution of 1024×768 while at the same time applying a brightening effect, follow these steps:

1. Check that **Run-time control of image size and clipping** is turned off in the **Parameter Settings: Resolution** page (see Figure 2–14 on page 2–21).

Figure 2–14. Resolution Parameter Settings for the Scaler



2. Set the image data format by choosing the input and output image width and height, the number of bits per pixel per color plane, and the number of color planes in sequence and parallel:

   - **Input image width**: `640`
   - **Input image height**: `480`
   - **Output image width**: `1,024`
   - **Output image height**: `768`
   - **Bits per pixel per color plane**: `8`
   - **Number of color planes**: `3`
   - **Color plane transmission format**: `Sequence`

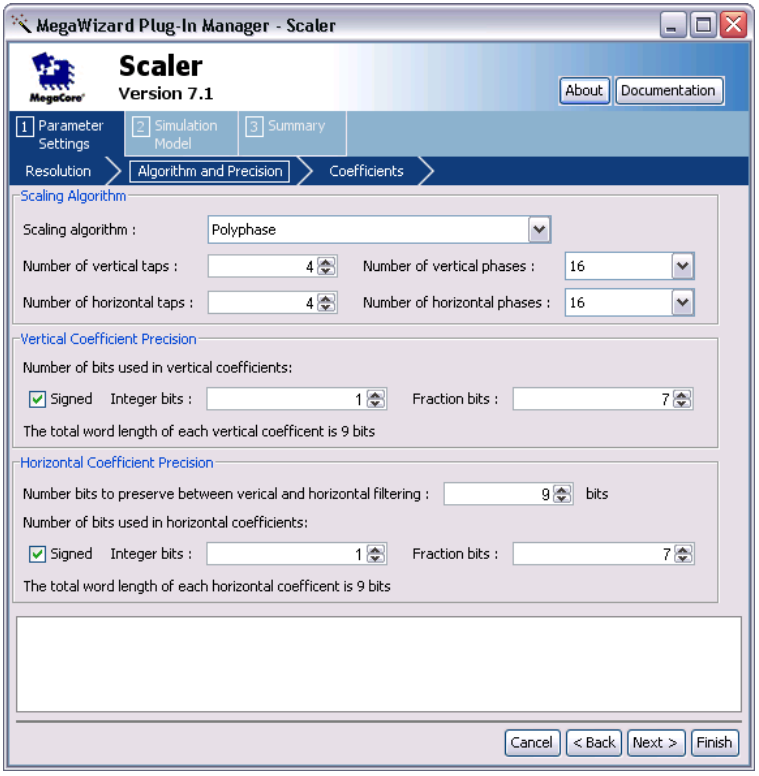
   The scaling direction must be the same in each dimension. Notice that if you pass through a state where the scaling is 640×768 to 1024×480, a warning message appears because horizontal upscaling is not supported at the same time as vertical downscaling.

3. Check that **Enable image clipping** is turned off.

See Table 4–20 on page 4–39 for more information about the options on the Scaler **Parameter Settings: Resolution** page.

4. Click **Next** to display the **Parameter Settings: Algorithm and Precision** page (Figure 2–15).

Figure 2–15. Algorithm and Precision Parameter Settings for the Scaler



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

   The scaling algorithm should default to `Polyphase`. All parameters on this page can be left at their default values for this tutorial. Notice that with signed coefficients, 1 integer bit, and 7 fraction bits, the total word length of each coefficient is 9 and that 9 bits are preserved between vertical and horizontal filtering. This means that with 4 vertical and 4 horizontal taps, the scaler uses a total of 8 9×9 DSP blocks.

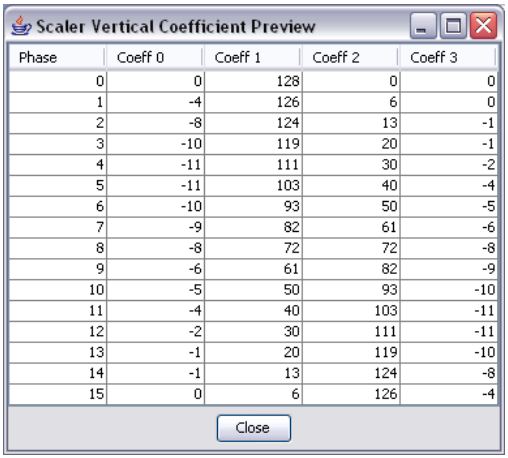
See Table 4–21 on page 4–40 for more information about the options on the Scaler **Parameter Settings: Algorithm and Precision** page.

6. Click **Next** to display the **Parameter Settings: Coefficients** page (Figure 2–16).

Figure 2–16. Coefficients Parameter Settings for the Scaler

7. Click **Preview coefficients** on the **Coefficients** page to view the quantized Lanczos 2 coefficients. (The default coefficients are the same for vertical and horizontal data, see Figure 2–17).

Figure 2–17. 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

8. Use Shift+click and Ctrl+c to select all these coefficients and copy them to the clipboard.

9. 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.

Each row of coefficients should be must sum to the same value See "Choosing and Loading Coefficients" on page 4–19.

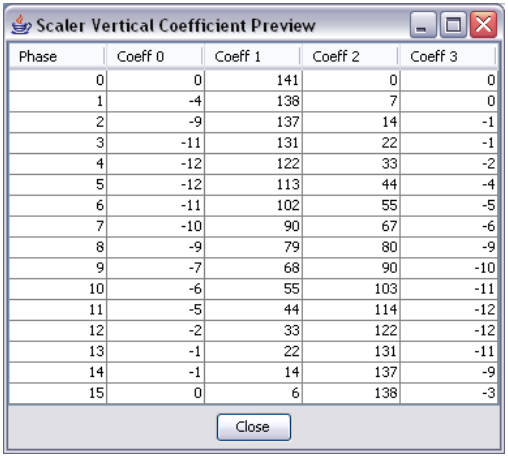
10. For each of the vertical and horizontal coefficient data panes in the **Coefficients** page:

    a. Set the **Filter function** to `Custom`.

    b. Click Browse, and select the CSV file that you created.

c. Click **Preview coefficients** to confirm that the data has been read correctly (see Figure 2–18).

Figure 2–18. 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

d. Perform these steps for both the vertical and horizontal coefficient data using the same CSV file.

See Table 4–22 on page 4–40 for more information about the options on the Scaler **Parameter Settings: Coefficients** page.

11. Click **Next** to complete the parameterization and display the **Simulation Model** page (Figure 2–21 on page 2–28).

Deinterlacer

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

1. Set the image data format in the **Parameter Settings** page (see Figure 2–19 on page 2–26) by choosing the image resolution, the number of bits per pixel per color plane, and the number of color planes that are received and transmitted in sequence:

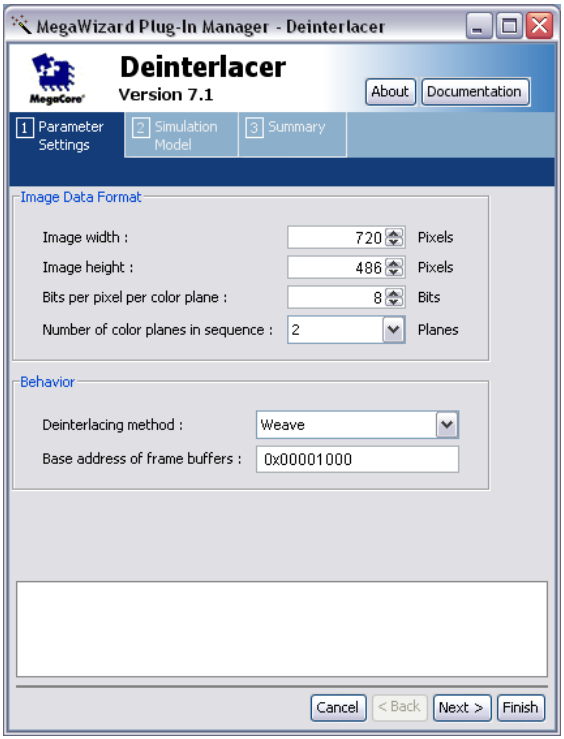
   - **Image width**: 720
   - **Image height**: 486
   - **Bits per pixel per color plane**: 8
   - **Number of color planes in sequence**: 2

2. Choose the `Weave` deinterlacing method.

   Notice that the option to specify the base address for the frame buffers is now available.

3. Specify the base address of frame buffers to be `0x00001000`.

Figure 2–19. Parameter Settings for the Deinterlacer



See Table 4–24 on page 4–42 for more information about the options on the Deinterlacer **Parameter Settings** page.

4. Click **Next** to complete the parameterization and display the **Simulation Model** page (Figure 2–21 on page 2–28).

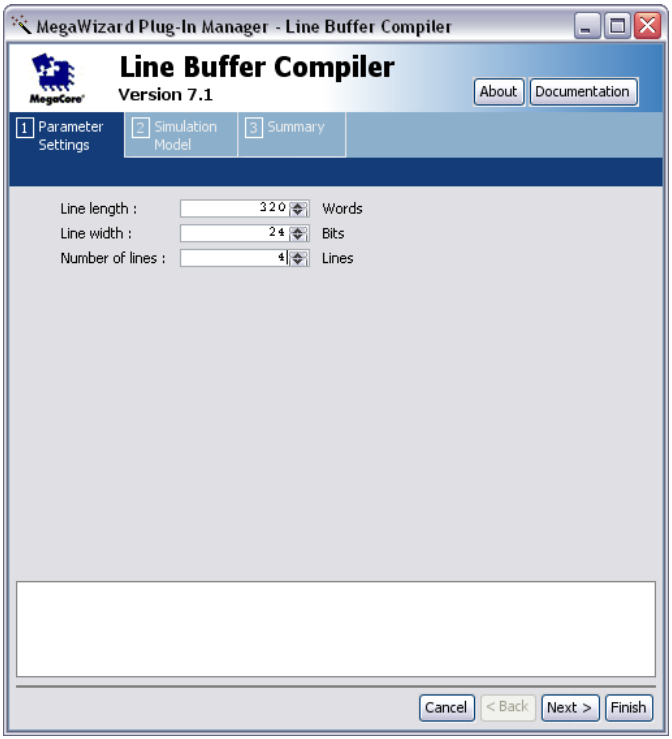
Line Buffer Compiler

To parameterize your Line Buffer Compiler function for a set of four line buffers each capable of holding 320 24-bit words, follow these steps:

1. Set the line length, line width and number of lines in the **Parameter Settings** page (see Figure 2–20):

   - **Line length**: 320
   - **Line width**: 24
   - **Number of lines**: 4

Figure 2–20. Parameter Settings for the Line Buffer Compiler



See Table 4–25 on page 4–42 for more information about these options.

2. Click **Next** to complete the parameterization and display the **Simulation Model** page (Figure 2–21 on page 2–28).

Set Up Simulation

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software. The model allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

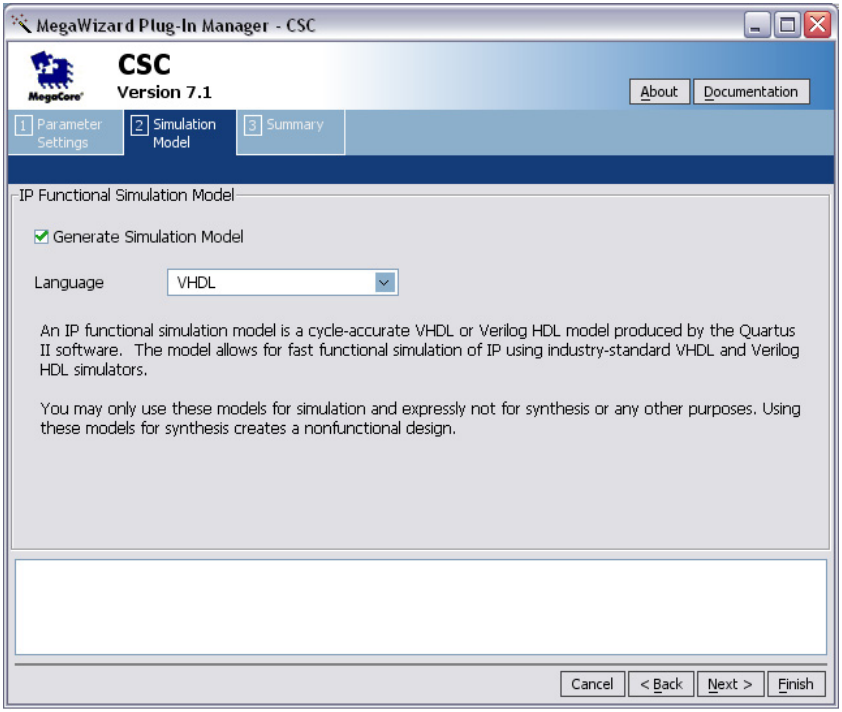
CAUTION

You may only use these models for simulation and expressly not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

To generate an IP functional simulation model for your MegaCore function, follow these steps:

1. Turn on **Generate Simulation Model** in the **Simulation Model** page. (Figure 2–21 shows the **Simulation Model** page for the Color Space Converter.)

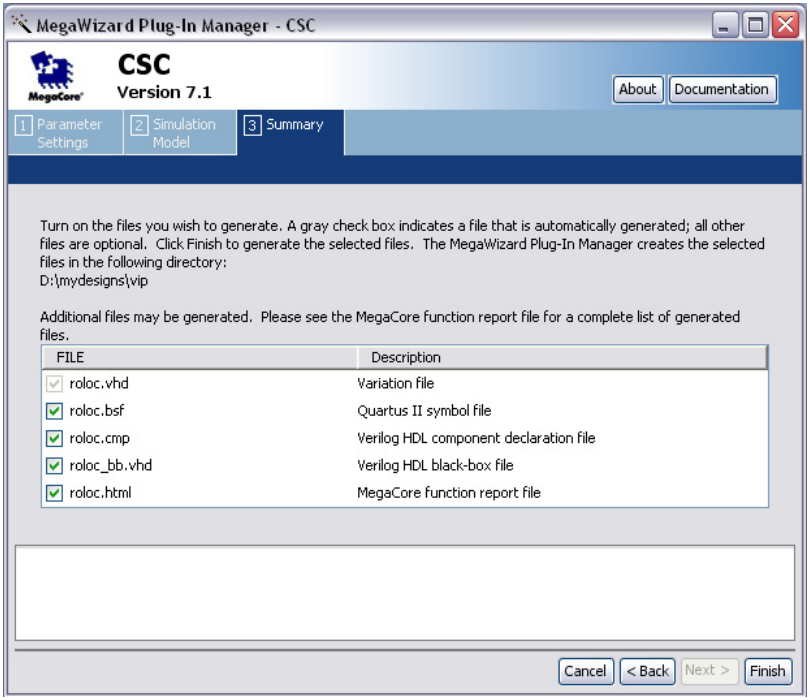
Figure 2–21. Simulation Model page for the Color Space Converter



2. Choose the required HDL language from the **Language** list.

3. Click **Next** to display the **Summary** page (Figure 2–22 shows the **Summary** page for the Color Space Converter.)

Figure 2–22. Summary Page for the Color Space Converter



Generate Files

You can use the check boxes on the **Summary** page to enable or disable the generation of specified files:

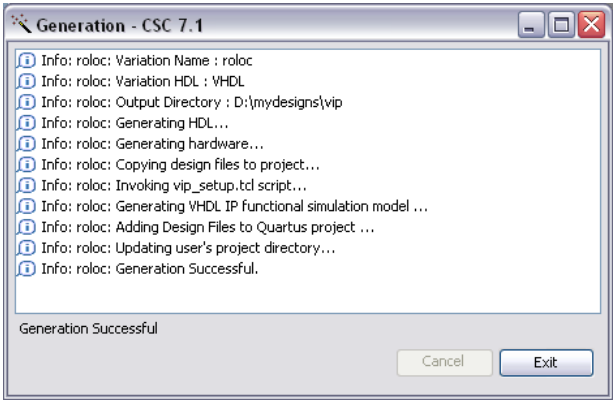
1. For this tutorial, turn on all of the check boxes.

   The variation file check box is always enabled and cannot be turned off. At this stage, you can still click **Back** to display the previous page or click **Parameter Settings**, **Simulation Model** or **Summary**, if you want to change any of the MegaWizard options.

2. To generate your MegaCore function and close the MegaWizard Plug-In manager, click **Finish**.

The generation phase may take several minutes to complete. The generation progress and status is displayed in a report window. (Figure 2–23 shows a typical generation report displayed for the Color Space Converter.)

Figure 2–23. Generation Report for the Color Space Converter



A MegaCore function report file containing a list of the design files and ports defined for your MegaCore function variation is saved as a HTML file if you turned on the **MegaCore function report file** check box in the **Summary** page.

3. Open the MegaCore function report file in your HTML browser.

   The file is saved at the location you specified in the MegaWizard Plug-In Manager and was also displayed in the **Summary** page before you generated the files. (For this tutorial, the specified location was: **d:\mydesigns\vip**.) Table 2–1 describes the generated files. The files in the report vary based on various design factors and parameters. 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), Note (2) (Part 1 of 2)

Filename	Description
<*entity name*>**.fsi**	Fast functional simulation information file.
<*entity name*>**.ocp**	Encyrypted OpenCore Plus file.
<*entity name*>**.vhd**	A hardware file that defines the design entity. This file is automatically generated when you click **Finish** in the MegaWizard Plug-in Manager and contains different hardware for each instance of the MegaCore function.

Table 2–1. Generated Files *Note (1)*, *Note (2)* ***(Part 2 of 2)***

Filename	Description
<*entity name*>_**setup.tcl**	Generated Tcl script that can be used to add the MegaCore function's design files to a Quartus II project.
<*entity name*>_***.hex**	Intel format HEX files for pre-loading on-chip memories used by the MegaCore function.
vip_setup.tcl	Top-level Tcl script that invokes <*entity name*>_**setup.tcl** within an open Quartus II project.
build_quartus.tcl	Generated Tcl script that can be used to create a fresh Quartus II project and compile the MegaCore function variation in the Quartus II software.
<*variation name*>**.bsf**	Quartus II 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*>.**html**	The MegaCore function variation report file.
<*variation name*>.**vhd**, or **.v**	A MegaCore function variation file that defines a VHDL or Verilog HDL 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 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.
alt_*.vhd, tta_x*.vhd,	VHDL component modules used by the MegaCore function.
hopt_*.vqm	Verilog Quartus Mapping (VQM) modules used by the MegaCore function.

Note to Table 2–1:

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

(2) The <*entity name*> prefix is added automatically. The VHDL code for each MegaCore instance is generated dynamically when you click **Finish** so that the <entity name> is different for every instance. It is generated from the <*variation name*> by appending an underscore and a three character code unique to the MegaCore function: Color Space Converter **_csc**, Chroma Resampler **_crs**, Gamma Corrector **_gam**, 2D FIR Filter **_fir**, 2D Median Filter **_med**, Alpha Blending Mixer **_mix**, Scaler **_scl**, Deinterlacer **_dil**, Line Buffer Compiler **_lbc**.

The MegaCore function report also lists the MegaCore function variation file ports. (Figure 2–24 shows a ports list for the Color Space Converter).

For a full description of the signals supported on external ports for your MegaCore function variation, refer to "Signals" on page 4–43.

Figure 2–24. Ports List in the MegaCore Function Report

MegaCore Function Variation File Ports

Name	Direction	Width
din_data	INPUT	8
din_ready	OUTPUT	1
din_valid	INPUT	1
dout_data	OUTPUT	8
dout_ready	INPUT	1
dout_valid	OUTPUT	1
clock	INPUT	1
reset	INPUT	1

4. After you have reviewed the generation report, click **Exit** to return to the Quartus II software.

You can now integrate your custom MegaCore function variation into your design.

Simulate the Design

You can simulate your design using the MegaWizard-generated VHDL and Verilog HDL IP functional simulation models. These are defined by the **.vo** or **.vho** file you specified in "Set Up Simulation" on page 2–28. 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*.

Video and Image Processing Suite MegaCore functions are usually simulated within DSP Builder.

For information about simulating MegaCore functions in DSP Builder, refer to the *DSP Builder User Guide*.

Compile the Design

You can use the Quartus II software to compile your design. Refer to Quartus II Help for instructions on compiling your design.

Program a Device

After you have compiled your design, program your targeted Altera device, and verify your design in hardware.

With Altera's free OpenCore Plus evaluation feature, you can evaluate a Video and Image Processing Suite MegaCore function before you purchase a license. OpenCore Plus evaluation allows you to generate an IP functional simulation model, and produce a time-limited programming file to evaluate your design in hardware.

For more information on OpenCore Plus hardware evaluation using the Video and Image Processing Suite MegaCore function, see "OpenCore Plus Time-Out Behavior" on page 4–30 and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Set Up Licensing

You need to 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 e-mails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative.

3. Interfaces

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 DSP Builder and SOPC Builder.

Three types of interface are used:

- Avalon® Streaming (Avalon-ST) interfaces pass serialized streams of pixel data into and out of the video MegaCore functions. A standard image streaming protocol defines how any type of video data can be broken down into Avalon-ST streams of pixel data. This is the main method provided for connecting the MegaCore functions together to form image processing datapaths.
- Avalon Memory-Mapped (Avalon-MM) slave interfaces provide run-time control input.
- Avalon-MM master interfaces are used where the MegaCore functions require external memory.

Refer to the *Avalon Streaming Interface Specification* and the *Avalon Memory-Mapped Interface Specification* for more information about these interface types.

These three types of interface cover all of the data input and output requirements for eight of the nine MegaCore functions in the Video and Image Processing Suite. The only exception is the Line Buffer Compiler MegaCore function which uses a lower level interface. For information about the interface used by this MegaCore function, refer to the functional description of the "Line Buffer Compiler" on page 4–24.

Avalon-ST Interfaces

Avalon-ST interfaces are used to pass video data into and out of the video MegaCore functions. A standard image streaming protocol is used to define how video frames can be broken down into serialized streams of data suitable for transmission according to the *Avalon Streaming Interface Specification*.

The image streaming protocol defines a mechanism for transmitting video data between the MegaCore functions on an FPGA which is sufficiently flexible to be useful across a wide range of video and image processing applications.

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.

The image streaming protocol defines a set of five parameters which are used to describe serialized streams of pixel data. These five parameters contain enough information to describe how any stream of pixel data conforming to the protocol should be reconstructed into a sequence of video images.

Table 3–1 lists the parameters and gives some examples of how they can be used.

Table 3–1. Examples of Image Streaming Protocol Parameters

Parameters					Description
Frame Width	*Frame Height*	*Interlaced / Progressive*	*Bits per Color Sample*	*Color Pattern*	
640	480	Progressive	8		640×480 pixel progressive scanned R'G'B' video where the 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.
1920	1080	Progressive	10		1920×1080 pixel progressive scanned R'G'B' video where the 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.
720	576	Interlaced	10		720×576 pixel interlaced video in the Y'CbCr color space, where there are twice as many Y samples as Cb or Cr samples and 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.

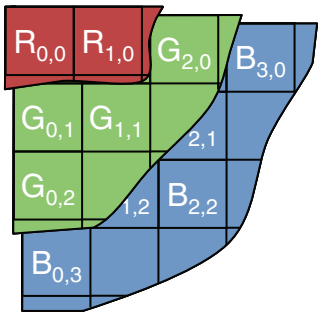
Examples

Consider a video sequence consisting of 640×480 pixel progressive frames 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 $R_{x,y}$ $G_{x,y}$ and $B_{x,y}$ 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 3–1 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 3–1. Part of a R'G'B' Frame



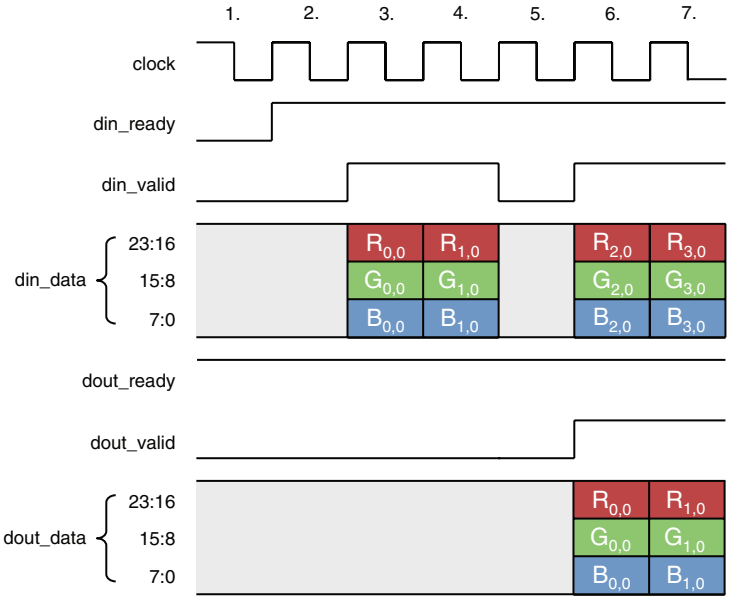
The number of binary bits used to represent each color sample varies between different video systems. In the following examples, assume that the video data has eight bits per pixel per color plane. This means that all of the $R_{x,y}$ $G_{x,y}$ and $B_{x,y}$ values for all x and y are eight bit values.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

The image streaming protocol can be used to describe different ways to transfer the same video data by 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 3–2 on page 3–4 shows a timing diagram illustrating how the first few pixels of a frame in the video format described above might be processed by a MegaCore function which handles R'G'B' in parallel.

Figure 3–2. 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 three signals (ready, valid and data) 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 which should be sampled. The three output port signals have equivalent but opposite semantics.

The sequence of events shown in Figure 3–2 is as follows:

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. See the "Specifications" on page 4–1 for further details of each MegaCore function.

2. The MegaCore function sets `din_ready` to logic '1', indicating that the input port will be 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 Streaming Interface Specification*. 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 puts all three color values of the top left pixel of the frame on to `din_data`.

4. The source holds `din_valid` at logic '1' and the pixel to the right of the first pixel is transferred on `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 then it could have become empty.

6. Data transmission resumes on the input port: `din_valid` transitions to logic '1' and the third 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 top left processed output pixel is transferred out three cycles after being input. See the "Specifications" on page 4–1 for guidelines about the latencies of each Video and Image Processing MegaCore function.

7. The fourth pixel is input and the second processed pixel is output.

In this example, both streams of pixel data have the image streaming protocol parameters shown in Table 3–2.

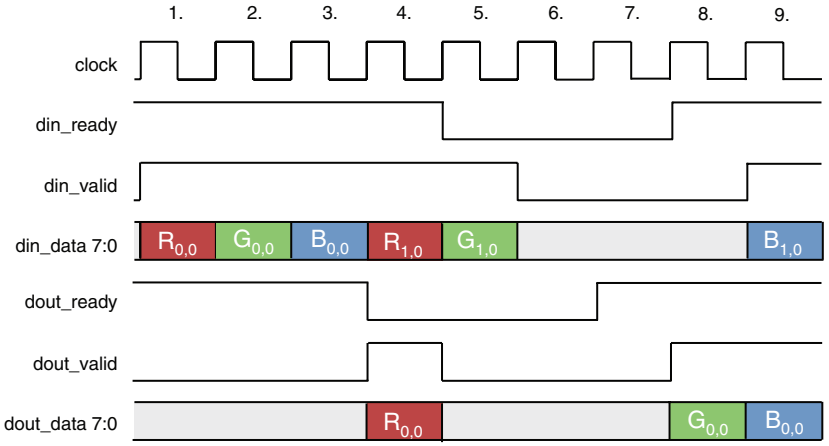
Table 3–2. Parameters for Example of Data Transferred in Parallel

Parameter	Value
Frame Width:	640
Frame Height:	480
Interlaced/Progressive:	Progressive
Bits per Color Sample:	8
Color Pattern:	

Data Transfer in Sequence

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

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



This example is similar to the previous example in all respects 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 3–3 is as follows:

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 of the top left pixel of the frame on the `din_data` port.
2. The source holds `din_valid` at logic '1' and the green color value of the top left pixel of the frame is input.
3. The corresponding blue color value 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 Streaming Interface Specification* states 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. 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 image streaming protocol parameters shown in Table 3–3.

Table 3–3. Parameters for Example of Data Transferred in Sequence

Parameter	Value
Frame Width:	640
Frame Height:	480
Interlaced/Progressive:	Progressive
Bits per Color Sample:	8
Color Pattern:	

Image Streaming Protocol Specification

The following specification is in three parts:

- The first part describes the parameters used by the image streaming protocol to describe a serialized stream of pixel data.
- The second part details the particular type of Avalon-ST interface used by the protocol and shows how that interface type depends upon the parameters of the data being transferred.
- The third part formally defines the rules governing how sequences of video images are serialized for transmission.

Parameters of the Image Streaming Protocol

Table 3–4 lists the parameters used by the image streaming protocol.

Table 3–4. Image Streaming Protocol Parameters

Parameter Name	Description	Example
Pixel Ordering 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 of the video stream. Note that in the case of interlaced video streams, height refers to the number of rows in each frame, not in each field. Must be a positive integer.	240
Interlaced/Progressive	"Interlaced" if this is an interlaced video stream, "Progressive" if it is progressive.	Progressive
Color Parameters:		
Bits per Color Sample	Maximum number of binary bits used to represent each color sample.	8
Color Pattern	A matrix that defines 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.	

A set of values for these parameters can be used to describe any type of of video data stream that can be transmitted according to the protocol. The example parameter values given in Table 3–4 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.

Specification of the Type of Avalon Streaming Interfaces Used

The *Avalon Streaming Interface* specification defines parameters which can be used to specify any type of Avalon-ST interface.

Table 3–5 on page 3–9 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 3–5. 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 Streaming Interface* specification defines many signal types many of which are optional.

Table 3–6 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 3–6. 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

Rules of the Image Streaming Protocol

This section specifies in detail the rules of the image streaming protocol. These rules, combined with a set of image streaming 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. The relationship between the image streaming protocol parameters and the type of video clip transmitted is described. A procedure is then shown for reconstructing a video clip from any image streaming protocol stream, given a set of parameter values. This procedure is the specification of the rules of the image streaming protocol.

A flexible definition of a video clip is required because there are many different types of video that can be transferred using the image streaming protocol. A video clip is defined to be 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. A field contains a set of color planes. For example, R, G and B color planes in full color R'G'B' video.

Because the image streaming protocol supports various kinds of subsampled video data, the color planes of a field are allowed to 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 image streaming protocol 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:

(1) $$Width_C = I_C \times R$$

where I_C is the number of times *C* appears in the *Color Pattern*, and *R* is equal to the *Frame Width* divided by the maximum value of I_C for all color planes. (The image streaming protocol specification requires that *R* be an integer.)

(2) $$Height_C = \frac{Frame\ \ Height}{Number\ \ of\ \ Fields}$$ if *C* is not vertically subsampled

(3) $$Height_C = \frac{Frame\ \ Height}{2 \times Number\ \ of\ \ Fields}$$ otherwise

where *Number of Fields* is the number of fields per frame (2 for interlaced, or 1 for progressive).

For example, consider a stream of pixel data with the parameters shown in Table 3–7.

Table 3–7. 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 being transmitted has the same three color planes. Y occurs twice in the pattern, Cb and Cr occur once each:

(4) $I_Y = 2;$

(5) $I_{Cb} = I_{Cr} = 1$

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

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

Using I_Y, I_{Cb}, I_{Cr} and *R* it is then possible to calculate the width of each color plane:

(7) $Width_Y = I_Y \times R = 2 \times 160 = 320$

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

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

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:

(10) $$Height_Y = \frac{Frame\ \ Height}{Number\ \ of\ \ Fields} = \frac{240}{1} = 240$$

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

$$Height_{Cb} = \frac{Frame\ \ Height}{2 \times Number\ \ of\ \ Fields} = \frac{240}{2} = 120 \quad (11)$$

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.

In the following 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-ST specification, symbol 0 occupies the most significant bits in the data word, symbol 1 is next and so on.

1. Firstly, input data is read and split it into a set of ordered sequences C_S, of samples for each color, *C* by 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) otherwise
                else
                    C ← plane(E)
                end if
                append D(j) to C_S
            end for
        end for
    end for
end for
```

2. Secondly, construct a video frame using the sequences of color samples C_S:

Frame ← create a new empty frame
  for ($0 \leq f <$ *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 $Width_C \times Height_C$
    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

Avalon-MM Slave Interfaces

The Video and Image Processing Suite MegaCore functions which 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. These registers should be assumed to 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, see the Parameters section for each MegaCore function in the "Specifications" chapter.

The first two registers of every control interface perform the same function as described below, 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 into 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 core begins processing. A few cycles after `Go` is set, the core begins processing data. If the `Go` bit is unset while the core is processing data, then the core will stop processing data again at the end of the current video frame, and wait 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 `Go` and `Status` registers can be used 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. Go to step 1.

The procedure above ensures that the update is performed exactly once per frame and that the core 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.

Specification of the Type of Avalon-MM Slave Interfaces Used

The *Avalon Memory-Mapped Interface Specification* defines many signal types, many of which are optional. Table 3–8 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 3–8. Avalon-MM Slave Interface Signal Types

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

Note that clock and reset signal types are not included. Video and Image Processing Suite v7.1 does not support Avalon-MM interfaces in multiple clock domains. Instead, all of the Video and Image Processing MegaCore functions have one clock input and one reset input. The Avalon-MM slave interfaces must operate synchronous to this clock.

The *Avalon Memory-Mapped Interface Specification* defines 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

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 Memory-Mapped Interface Specification* defines many signal types, many of which are optional. Table 3–9 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 3–9. Avalon-MM Master Interface Signal Types

Signal	Width	Direction	Usage
`waitrequest`	1	Input	Read-write
`address`	32	Output	Read-write
`read`	1	Output	Read-only
`readdata`	64	Input	Read-only
`write`	1	Output	Write-only
`writedata`	64	Output	Write-only
`readdatavalid`	1	Input	Read-only

Note that clock and reset signal types are not included. Video and Image Processing Suite v7.1 does not support Avalon-MM interfaces in multiple clock domains. Instead, all of the Video and Image Processing MegaCore functions have one clock input and one reset input. The Avalon-MM master interfaces must operate synchronous to this clock.

Some of the signals in Table 3–9 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 Memory-Mapped Interface Specification* defines 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

4. Specifications

Functional Description

Each Video and Image Processing MegaCore function is implemented such that it generates hardware to perform its operation 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 input guard bands can reduce the resource usage of the MegaCore function, and 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 9 constant coefficients and 3 constant summands that relate the color spaces.

Given a set of 9 constant coefficients [*A0*, *A1*, *A2*, *B0*, *B1*, *B2*, *C0*, *C1*, *C2*] and a set of 3 constant 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:

(1) $dout\_0 = (A0 \times din\_0) + (B0 \times din\_1) + (C0 \times din\_2) + S0$

(2) $dout\_1 = (A1 \times din\_0) + (B1 \times din\_1) + (C1 \times din\_2) + S1$

(3) $dout\_2 = (A2 \times din\_0) + (B2 \times din\_1) + (C2 \times din\_2) + S2$

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 R'G'B' to Y'CbCr: SDTV
- Y'CbCr: SDTV to Computer R'G'B'
- Computer R'G'B' to Y'CbCr: HDTV
- Y'CbCr: HDTV to Computer R'G'B'
- Studio R'G'B' to Y'CbCr: SDTV
- Y'CbCr: SDTV to Studio R'G'B'
- Studio R'G'B to Y'CbCr: HDTV
- Y'CbCr: HDTV to Studio R'G'B'
- Y'IQ to Computer R'G'B'
- Computer R'G'B' to Y'IQ
- Y'UV to Computer R'G'B'
- Computer R'G'B' to Y'UV

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

Predefined conversions only support unsigned input and output data. If signed input or output data is selected, the predefined conversion will produce incorrect results. When using a predefined conversion, the precision of the constants must still be defined.

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 dialog box matrix) are rounded to fit into the chosen fixed point type (these are shown in the purple cells of the dialog box matrix).

Calculation Precision

The Color Space Converter MegaCore function does not loose calculation precision during the conversion. The calculation and result data types are derived from: the range of 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 to Output Data Type Conversion

After the calculation, it is necessary to convert the fixed point type of the results to the integer data type of the output. This is performed in four stages, in the following order:

1. **Result Scaling**. The results can be scaled up, increasing their range. This is useful to quickly increase the color depth of the output. The available options are powers of 2, from 1 to 64. This is implemented as a simple shift operation so it does not require multipliers.

2. **Removal of Fractional Bits**. If any fractional bits exist, this option becomes available. There are two methods:

    a. Truncation of fractional bits. Fractional bits are removed from the data. This is equivalent to rounding towards negative infinity.

    b. Rounding to nearest integer. Fractional bits are removed, rounding to the closest integer. If the fractional bits are equal to 0.5 then rounding is towards positive infinity.

3. **Conversion from Signed to Unsigned**. If any negative numbers can exist in the results and the output type is unsigned, then this option becomes available. There are three methods:

    a. Saturate negative numbers to 0.

    b. Convert negative numbers to their absolute positive value (negate).

c. Ignore the possibility. This option treats signed 2's complement numbers as unsigned numbers; that is, in 8-bit output, -1 would equal 255.

4. **Constrain to Range**. If any of the results are beyond the range specified by the output data type, this option becomes available. There are two methods:

   a. Saturate to the minimum and maximum values allowed by the specified range (output guard bands, or if unspecified the minimum and maximum values allowed by the output bits per pixel).

   b. Ignore the possibility. The output is the least significant bits of the result that will fit into the selected output type. If output guard bands are selected, they are not used.

Ignoring the possibility (3c and 4b) can be useful if within the input color space not all combinations of channel values are legal colors. With this information, it may be known that the actual range of results is less than the Color Space Converter MegaCore function calculates, and as such the logic to constrain to the output data type, or convert signed to unsigned is unnecessary.

The Color Space Converter MegaCore function can process streams of pixel data of the types shown in Table 4–1.

Table 4–1. Color Space Converter Image Streaming Protocol Parameters

Parameter	Value
Frame Width:	As selected in the MegaWizard interface.
Frame Height:	As selected in the MegaWizard interface.
Interlaced/Progressive:	Either.
Bits per Color Sample:	As selected in the MegaWizard interface.
Color Pattern:	For color planes in sequence:  For color planes in parallel: 

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:

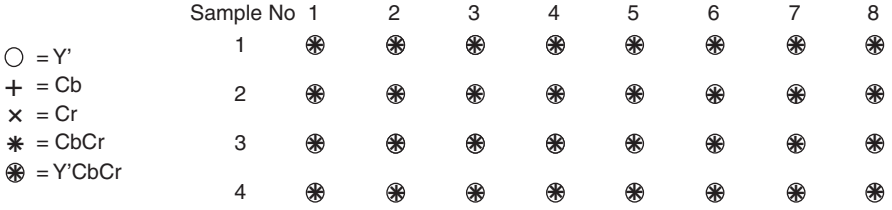
- 4:4:4 specifies full resolution in planes 1, 2, and 3
- 4:2:2 specifies full resolution in plane 1; half width resolution in planes 2 and 3
- 4:2:0 specifies full resolution in plane 1; half width and height resolution in planes 2 &3

One of the input or output sampling rates must be 4:4:4.

Independent control of the horizontal and vertical format conversion methods is provided. For each, you may choose between no interpolation (nearest neighbor pixel) or linear interpolation.

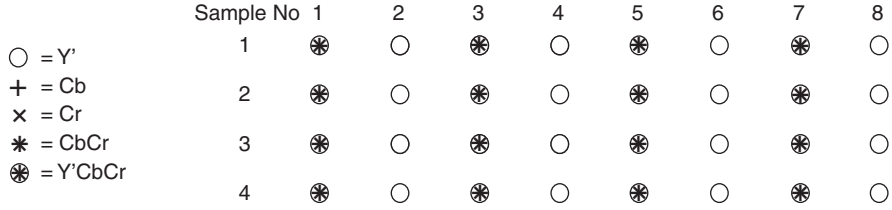
4:4:4 sampling of Y'CbCr is represented diagrammatically in Figure 4–1.

Figure 4–1. Y'CbCr with 4.4.4 Sampling Rate



For resampling between 4:4:4 and 4:2:2 with no interpolation, every other sample in the horizontal planes of Cb, Cr are dropped as shown in Figure 4–2.

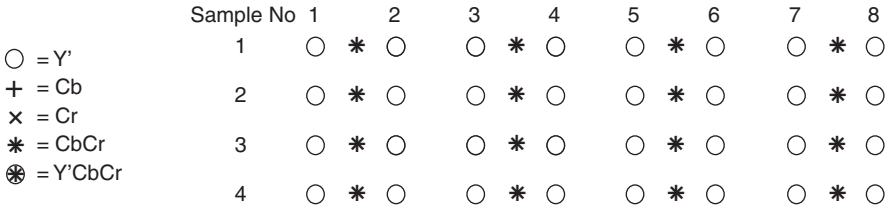
Figure 4–2. Resampling 4.4.4 to 4.2.2 Without Interpolation



The data ports of the Chroma Resampler MegaCore expect 4:2:2 data to be transmitted in the order Y' Cb Y' Cr. Each row of an image that is *W* pixels wide will therefore be represented by *W*/2 repetitions of the pattern Y' Cb Y' Cr.

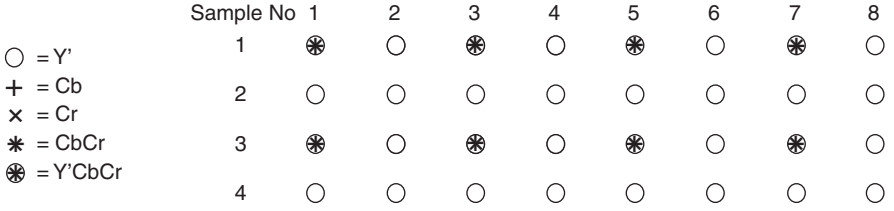
With linear interpolation, each pair of values in the horizontal planes Cb, Cr are averaged to provide calculated values between the originals as shown in Figure 4–3.

Figure 4–3. Resampling 4.4.4 to 4.2.2 With Linear Interpolation



For resampling between 4:4:4 and 4:2:0 with no interpolation, 4:2:0 samples are arranged as shown in Figure 4–4.

Figure 4–4. Resampling 4.4.4 to 4.2.0 Without Interpolation

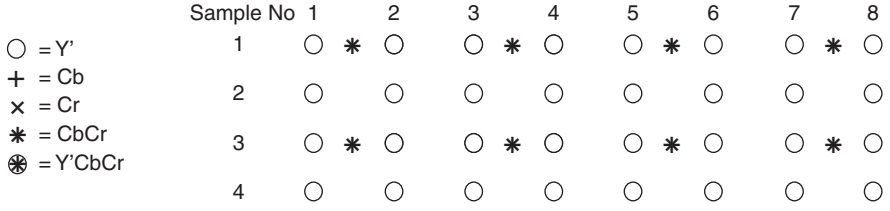


The data ports of the Chroma Resampler MegaCore expect 4:2:0 data to be transmitted in the order Y'(Cb+Cr)Y'; that is, a Y', followed by a Cb or a Cr, followed by another Y'.

On even rows of an image represented in this way, the middle sample of this pattern will be a Cb. On odd rows, the middle sample will be a Cr. Each row of an image that is *W* pixels wide will therefore be represented by *W*/2 repetitions of the pattern Y'(Cb+Cr)Y'. An entire frame of data for an image that is *H* pixels high will consist of *H* such rows.

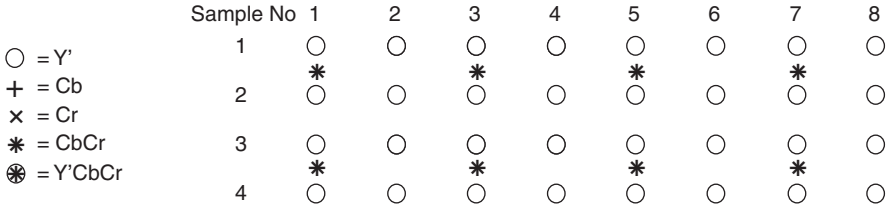
With horizontal interpolation, the samples are arranged as shown in Figure 4–5.

Figure 4–5. Resampling 4.4.4 to 4.2.0 With Horizontal Interpolation



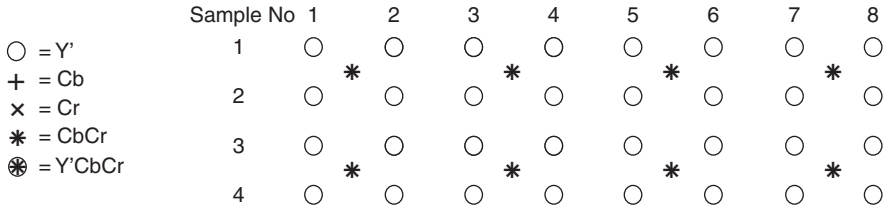
With vertical interpolation (MPEG-2 standard), they are arranged as shown in Figure 4–6.

Figure 4–6. Resampling 4.4.4 to 4.2.0 With Vertical Interpolation



With both horizontal and vertical interpolation (MPEG-1, H.261, and H.263 standards), they are arranged as shown in Figure 4–7.

Figure 4–7. Resampling 4.4.4 to 4.2.0 With Horizontal & Vertical Interpolation



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 Chroma Resampler MegaCore function are also in the same unsigned format.

The Chroma Resampler MegaCore function can process streams of pixel data of the types shown in Table 4–2.

Table 4–2. Chroma Resampler Image Streaming 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:	As selected in the MegaWizard interface.
Color Pattern:	For 4:4:4 data:  For 4:2:2 data:  For 4:2:0 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 into the LUT by external hardware using this interface. See Table 4–28 on page 4–44 for information about the control signals used for this interface.

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 0 and 1 are the Go and Status registers. These can be used to stop and restart the Gamma Corrector MegaCore function. All Video and Image Processing MegaCore functions which have control interfaces provide Go and Status registers which operate in exactly the same way. For details of the register map for this control port, see Table 4–14 on page 4–34.

For general 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 3–13.

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 $x + 2$. For full details of the register map for this control port, see Table 4–14 on page 4–34.

The Gamma Corrector MegaCore function can process streams of pixel data of the types shown in Table 4–3.

Table 4–3. Gamma Corrector Image Streaming Protocol Parameters

Parameter	Value
Frame Width:	As selected in the MegaWizard interface.
Frame Height:	As selected in the MegaWizard interface.
Interlaced/Progressive:	Either.
Bits per Color Sample:	As selected in the MegaWizard interface.
Color Pattern:	One, two or three channels in sequence. For example if three channels in sequence is selected:  where α, β and γ 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 constant coefficients. The core retains full precision throughout the calculation, while making efficient use of FPGA resources. With suitable coefficients, the core 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 0's 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 also supports symmetric coefficients. This reduces the number of multipliers, resulting in smaller hardware.

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 in white boxes in the GUI) are rounded to fit into the chosen coefficient fixed point type (shown in the purple boxes of the GUI).

Scaling of the Result

After the FIR calculation, the result can be scaled up, and used to provide higher precision value for an existing coefficient set by choosing a wider output range. The available options are powers of 2, from 1 to 64. This is implemented as a shift operation so it does not require multipliers.

Data Type Conversion for Output

After the scaling, it may be necessary to lose result precision and/or range to match the selected output data type. Conversion to the selected output type is performed in three stages in the following order:

1. *Removal of fractional bits*. If any fractional bits exist, this option becomes available. There are two methods:
    a. Truncation of fractional bits. Fractional bits are removed from the data. This is equivalent to rounding towards negative infinity.
    b. Rounding to nearest integer. Fractional bits are removed, rounding to the closest integer. If the fractional bits are equal to 0.5 then rounding is towards positive infinity.
2. *Converting from signed to unsigned*. If the result of the FIR calculation is signed and the output type is unsigned, then this option becomes available. There are three methods:
    a. Saturate negative numbers to 0.

b. Convert negative numbers to their absolute positive value (negate).

c. Ignore the possibility. This will treat signed 2's complement numbers as unsigned numbers; that is, in 8-bit output, -1 would equal 255.

3. *Constrain to range*. If the result of the FIR is beyond the output range specified by the output guard bands, or the maximum and minimum for the given output data type, this option becomes available. There are two methods:

a. Saturate to the minimum and maximum values allowed by the specified range (output guard bands, or if unspecified the minimum and maximum of the output data type).

b. Ignore the possibility. The output is the least significant bits of the result that will fit into the selected output type. If output guard bands are selected, they are not used.

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

Table 4–4. 2D FIR Filter Image Streaming 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:	As selected in the MegaWizard interface.
Color Pattern:	One, two or three channels in sequence. For example if three channels in sequence is selected:  where α, β and γ 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 4–5.

Table 4–5. 2D Median Filter Image Streaming 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:	As selected in the MegaWizard interface.
Color Pattern:	One, two or three channels in sequence. For example if three channels in sequence is selected:  where α, β and γ can be any color plane.

Alpha Blending Mixer

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

The location of each layer can be changed dynamically when the core is running, and individual layers can be switched on and off. This run-time control is provided by an Avalon-MM slave port with registers for the location and on/off status of each layer. For details of the register map for this control port, see Table 4–19 on page 4–38.

Control data is read once 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.

For general 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 3–13.

The number of image layers mixed and the size of each layer 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, 1], where 1 represents full translucence, and 0 represents fully opaque. For *n*-bit alpha values (RGBAn) there is a range of [0, 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*, 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 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 4–6.

Table 4–6. Alpha Blending Mixer Image Streaming 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:	As selected in the MegaWizard interface (specified separately for image data and alpha blending).
Color Pattern (din & dout):	One, two or three channels in sequence. For example if three channels in sequence is selected:  where α, β and γ 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 and/or clip video streams. It supports Nearest Neighbor, Bilinear, Bicubic and Polyphase scaling algorithms.

The Scaler MegaCore function can be configured to change resolutions and/or filter coefficients at runtime using an Avalon-MM Slave interface.

For general 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 3–13.

In the formal definitions of the scaling algorithms given here, 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, it requires no DSP blocks, and uses less logic area 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.

Because this equation gives an answer relative to the mid-point of the input pixel, we should subtract 0.5 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 core is equivalent to the following integer calculation:

$$O(i, j) = F((2 \times w_{in} \times i + w_{in})/(2 \times w_{out}), (2 \times h_{in} \times j + h_{in})/(2 \times h_{out})) \quad (4)$$

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 of these is either the sum of the horizontal and vertical fraction bits plus 2, or the input data bit width, whichever is greater. For example, with 4 horizontal fraction bits, 3 vertical fraction bits, and 8 bit input data, the multipliers are 9 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 these 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:

(5) $$in_i = (i \times w_{in})/w_{out}$$

(6) $$in_j = (j \times h_{in})/h_{out}$$

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:

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

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

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 since they measure how far the real value is 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:

(9)

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

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 4 vertical and 4 horizontal taps. In the following discussion, all comments relating to the polyphase algorithm are applicable to the bicubic algorithm assuming 4×4 taps.

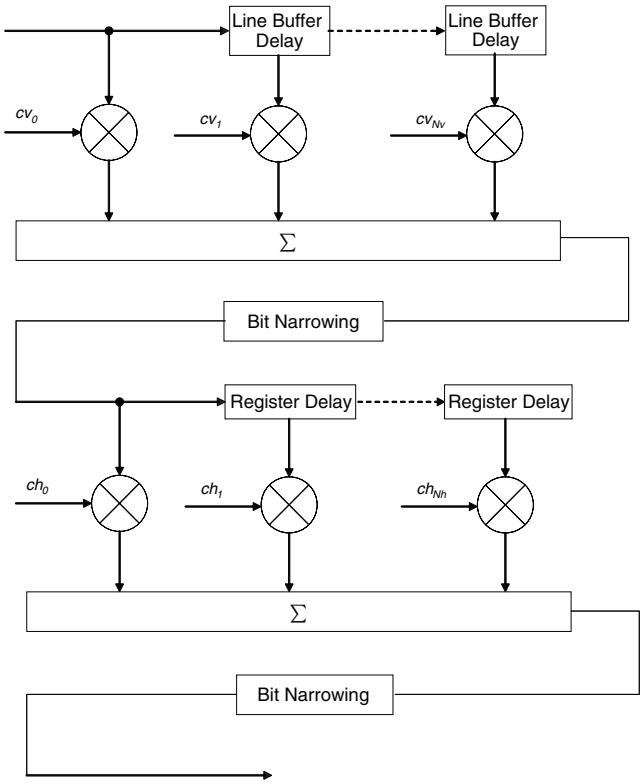
Figure 4–8 on page 4–17 shows the flow of data through an instance of the scaler in polyphase mode.

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.

Note that the progress of data through the taps (line buffer and register delays) and the values of the coefficients used in the multiplication are controlled by logic that is not present in the diagram. This logic is described in "Algorithmic Description" on page 4–18.

Figure 4–8. 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.

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. See the "Number of bits to preserve between vertical and horizontal filtering" parameter in Table 4–21 on page 4–40.

The memory requirement is N_v line-buffers plus a vertical and a horizontal coefficient bank. 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 bank is $N_v \times B_v$ bits wide and has P_v entries. This bank is stored in memories that will each be no more than 64 bits wide and the same depth as the bank. If the bank is more than 64 bits wide, it is mapped to multiple memories. The Quartus II software will map each memory to either the on-chip RAM of the FPGA or to logic elements in the FPGA.

The horizontal coefficients are handled similarly, using the corresponding horizontal parameters.

If the horizontal and vertical coefficients are identical, they are stored in the same memory.

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. In the case of up scaling, this `sinc` function performs interpolation. In the case of down scaling, it acts as a low-pass filter in order to remove high-frequency data that would cause aliasing in the smaller output image.

During the filtering process, the center of the `sinc` function should be at the center 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, these correspond to an $N_v \times N_h$ 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 centre of the kernel is placed at $(\lfloor in_i \rfloor, \lfloor in_j \rfloor)$ where in_i and in_j are calculated using equations 5 and 6 on page 4–15.

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:

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

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

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 into 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.

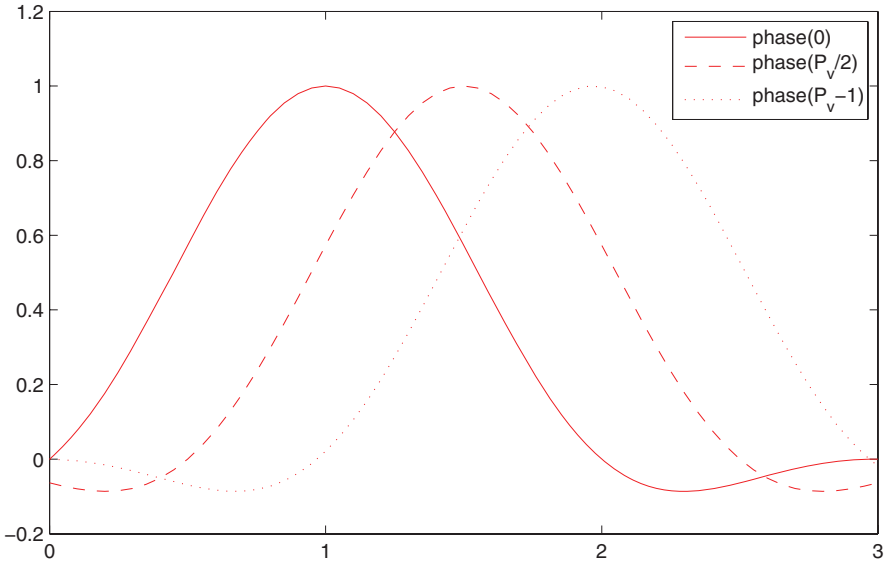
Choosing and Loading Coefficients

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 (see Table 4–23 on page 4–41). When the coefficients are read at runtime, they are checked once per frame and double-buffered so that they can be updated as the core processes active data without causing corruption.

Figure 4–9 on page 4–20 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.

Figure 4–9. Lanczos 2 Function at Various Phases



The class of Lanczos N functions is defined as:

(12)

$$LanczosN(x) = \begin{cases} 1 & x = 0 \\ \frac{\sin(\pi x)}{\pi x}\frac{\sin(\pi x/N)}{\pi x/N} & x \neq 0 \wedge |x| < N \\ 0 & |x| \geq N \end{cases}$$

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 centre of the function in $1/P_v$ increments towards tap 2. The filtering coefficients applied in a 4-tap scaler for a particular phase are samples of where the function with that phase crosses 0, 1, 2, 3 on the x-axis.

The preset filtering functions are always spread over the number of taps given. For example, Lanczos 2 is defined over the range –2 to +2, but with 8 taps the coefficients are shifted and spread to cover 0 to 7.

Compile-time custom coefficients are loaded from a CSV file. One CSV file is specified for vertical coefficients and one for horizontal coefficients. For N taps and P phases, the file should contain $N \times P$ values. The values should be listed as N taps in order for phase 0, N taps for phase 1, up to the Nth tap of the Pth phase. They need not be presented with each phase on a separate line.

The values must be pre-quantized into the range implied by the number of integer, fraction, and sign bits chosen in the MegaWizard interface and have their fraction part multiplied out. The sum of any two coefficients in the same phase must also be within 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, a set of coefficients for an N-tap, P-phase instance of the scaler can be generated 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 load them in 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.*

This method does not follow the steps above, but instead obtains weights to use for each of the 4 taps in order to sample a cubic function that runs between tap 1 and tap 2 at a position equal to the phase variable described above. A consequence of this is that the bicubic coefficients are good for up scaling, but not for down scaling.

If the coefficients are "symmetric" and provided at compile-time, then only half the number of phases are stored. For N taps and P phases, an array, $C[P][N]$, of quantized coefficients is symmetric if for all $p \in [1, P-1]$ and all $t \in [0, N-1]$, $C[p][t] = C[P-p][N-1-t]$, that is phase 1 is phase $P-1$ with the taps in reverse order, phase 2 is phase $P-2$ reversed and so on. The predefined Lanczos and bicubic coefficient sets satisfy this property.

Recommended Parameters

In Polyphase mode, the parameters for the Scaler Megacore function must be chosen 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 4–7 summarizes some recommended values for parameters when using the scaler in polyphase mode.

Table 4–7. 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 4–8.

Table 4–8. Scaler Image Streaming 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:	As 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 α, β and γ can be any color plane.

Deinterlacer

The Deinterlacer MegaCore function provides a flexible and efficient means to convert interlaced video to progressive video using bob and weave algorithms.

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 Deinterlacer MegaCore function are also in the same unsigned format.

Deinterlacing Methods

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

Bob deinterlacing scales input fields up by a factor of two vertically. This function supports two types of scaling for bob deinterlacing: scanline duplication and scanline interpolation.

Bob Scanline Duplication
Scanline duplication simply scales by repeating each scanline in input field *0* twice to make the output frame. Input field *1* is discarded.

Bob Scanline Interpolation
Scanline interpolation recreates the lines missing from input field *0* by performing an unweighted mean of the lines above and below them. Input field *1* is discarded. At the bottom of field *0* there is only one line available, so this line is just duplicated as per scanline duplication.

Output Frame Rate

The Deinterlacer MegaCore function produces a *N*/2 Hz output frame rate for an *N*Hz input field rate. For example, 1080i @ 60Hz to 1080p @ 30Hz.

Triple Buffering

Weave deinterlacing requires a frame buffer stored in off-chip memory so that lines from different fields can be woven together. For this reason, the weave deinterlacer has a built in triple-buffering function.

When in weave mode, the deinterlacer has two 64-bit Avalon-MM master ports. These must be connected to an external memory with enough space to store three full frames of video data.

One way to do this is to connect the Deinterlacer MegaCore function to an Altera (double data rate) DDR Controller MegaCore function using SOPC Builder. The address in the Avalon-MM address space where the base of the frame buffer memory is to be located can be set when parameterizing the Deinterlacer MegaCore function.

The Deinterlacer MegaCore function can process streams of pixel data of the types shown in Table 4–9.

Table 4–9. Deinterlacer Image Streaming Protocol Parameters

Parameter	Value
Frame Width:	As selected in the MegaWizard interface.
Frame Height:	As selected in the MegaWizard interface.
Interlaced/Progressive:	Interlaced input, Progressive output.
Bits per Color Sample:	As selected in the MegaWizard interface.
Color Pattern:	one, two or three channels in sequence. For example if three channels in sequence is selected:  where α, β and γ can be any color plane.

Line Buffer Compiler

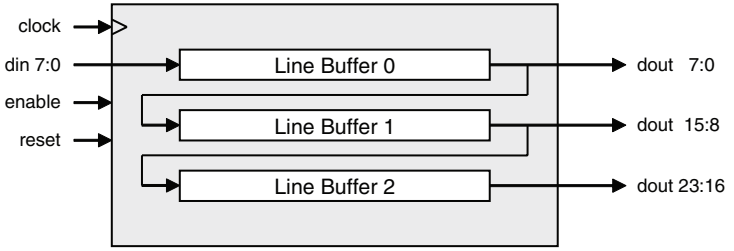
FPGA memory is a valuable resource for many video and imaging applications particularly when developing systems that require high definition resolutions and high order accuracy algorithms.

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

An example of the logic structure produced by the Line Buffer Compiler is shown in Figure 4–10 on page 4–25.

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.

Figure 4–10. Example of the Line Buffer Compiler Logic

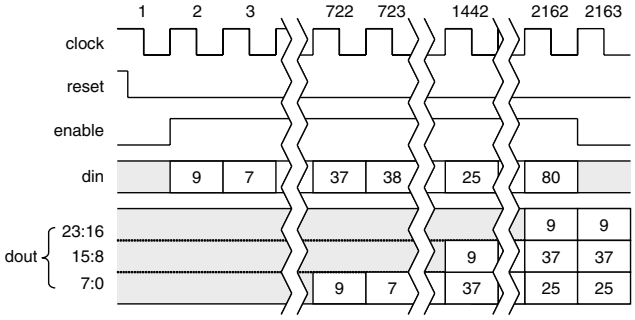


When `enable` is asserted, data flows into the module through the `din` port and passes through each of the line buffers in sequence. The output of all of the line buffers is concatenated together into a bus of size (*number of line buffers*) × (*line buffer width*) bits which can be read at any time.

Unlike the other MegaCores in the Video and Image Processing Suite, the Line Buffer Compiler does not provide an image streaming protocol based data interface.

Figure 4–11 on page 4–25 shows a timing diagram illustrating how a Line Buffer Compiler MegaCore function such as the one shown in Figure 4–10 would process data if the length of each line was set to 720 pixels.

Figure 4–11. Timing diagram Illustrating a Line Buffer Compiler in Use



The sequence of events shown in Figure 4–11 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 number `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 number `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 number `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 number `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 number `25` is driven on to the input.

2162. 2160 enabled clock cycles after the number `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`.

Stall Behavior

During data processing, the Video and Image Processing Suite MegaCore functions sometimes stall to allow extra time for internal processing. Typically, this occurs between rows of image data and between frames.

When stalled, the MegaCore function signals that it is not ready to consume 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 will stall, and thus not output data. With the exception of the Deinterlacer in `Weave` mode, which uses triple-buffering, 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.

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.

Chroma Resampler

All modes of the Chroma Resampler stall for a few cycles between frames. The MegaCore function will pause between rows when up sampling with horizontal interpolation. In all other operation modes, there is no stall in between image rows.

Latency from input to output varies depending on the operation mode of the Chroma Resampler MegaCore function, because varying numbers of internal line buffers are required:

- When down sampling from 4:4:4 to 4:2:0 or up sampling from 4:2:0 to 4:4:4 without vertical interpolation, one line buffer is used so there is a delay from input to output of a little more than one line of input data.
- When up sampling from 4:2:0 to 4:4:4 with vertical interpolation, two line buffers are required and the latency is roughly doubled.

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 fully sampled side on each clock cycle. The subsampled side will pause 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.

Gamma Corrector

In all parameterizations, the Gamma Corrector only stalls 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.

2D FIR Filter

There is a delay of a little more than $\lfloor N/2 \rfloor - 1$ 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.

2D Median Filter

There is a delay of a little more than $\lfloor N/2 \rfloor - 1$ 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.

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

When clipping is enabled, the Scaler produces no output during the time that it is throwing away pixels outside the clipping area. Clipped data is discarded at a rate of one sample per clock regardless of the scaling ratio.

When the reads are inside the clipping area, 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.

Within 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 listed above 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 $20 + 2(16 \times 32) = 1024$ cycles.

Deinterlacer

In Bob mode, the Deinterlacer processes each video frame in two stages:

- In the first stage, field zero is received on the input port. The Deinterlacer 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.
- In the second stage, field one is received on the input port and discarded, and no output is generated.

In Weave mode, data input and output are decoupled through the use of a triple buffer mechanism in external memory.

OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation supports the following two 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 may be masked by the time-out behavior of the other megafunctions.

The untethered timeout for all Video and Image Processing Suite MegaCore functions is 1 hour; the tethered timeout 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.

For more information on OpenCore Plus hardware evaluation, see "OpenCore Plus Evaluation" on page 1–5 and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Parameters

Tables 4–10 to 4–25 show the Video and Image Processing Suite MegaCore function parameters.

The default parameter values are shown using bold text in the tables.

Color Space Converter

Table 4–10 and Table 4–11 show the Color Space Converter MegaCore function parameters. These parameters can be set in the MegaWizard interface (see page 2–6).

Table 4–10. Color Space Converter Parameter Settings: General (Part 1 of 2)

Parameter	Value	Description
Image width	32–1920 Default = **640**	Choose the required image width in pixels.
Image height	32–1080 Default = **480**	Choose the required image height in pixels.

Table 4–10. Color Space Converter Parameter Settings: General (Part 2 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.
Multiply results by	**1**,2,4,8,16,32,64	Specify the scale factor for the results.
Remove fraction bits by	**Round values to nearest integer**, Truncate values to integer	Choose the method of discarding fraction bits resulting from the calculation.
Convert from signed to unsigned by	**Replacing negative values with zero**, Replacing negative with absolute value, Ignore negative value	Choose the method of signed to unsigned conversion for the results.
Constrain to output range by	**Saturating to min and max values**, Ignore range overflow and underflow	Choose the method used to constrain the output to a range.

Table 4–11. Color Space Converter Parameter Settings: Operands

Parameter	Value	Description
Color model conversion	Computer R'G'B' to Y'CbCr: SDTV Y'CbCr: SDTV to Computer R'G'B' Computer R'G'B' to Y'CbCr: HDTV Y'CbCr: HDTV to Computer R'G'B' Studio R'G'B' to Y'CbCr: SDTV Y'CbCr: SDTV to Studio R'G'B' Studio R'G'B to Y'CbCr: HDTV Y'CbCr: HDTV to Studio R'G'B' Y'IQ to Computer R'G'B' Computer R'G'B' to Y'IQ Y'UV to Computer R'G'B' Computer R'G'B' to Y'UV **Custom**	You can choose 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 choose **Computer R'G'B' to Y'CbCr: SDTV**, then *din_0* = R', *din_1* = B', *din_2* = G', *dout_0* = Y', *dout_1* = Cb, and *dout_2* = Cr.
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 4–12 shows the Chroma Resampler MegaCore function parameters. These parameters can be set in the MegaWizard interface (see page 2–10).

Table 4–12. Chroma Resampler Parameter Settings

Parameter	Value	Description
Image width	32–1920 Default = **640**	Choose the required image width in pixels.
Image height	32–1080 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).
Color Plane Configuration	Three color planes in sequence	There must always be three color planes in sequence for this function.
Conversion Format:	**4:4:4 to 4:2:2**, 4:4:4 to 4:2:0, 4:2:2 to 4:4:4, 4:2:0 to 4:4:4	Choose the format/sampling rate format for the input and output frames. Note that either the input or the output format must be 4:4:4.
Horizontal Interpolation:	**Linear**, Nearest Neighbor	Choose the interpolation method to use in the horizontal direction when re-sampling 4:4:4 data to or from 4:2:2 or 4:2:0.
Vertical Interpolation:	**Linear**, Nearest Neighbor	Choose the interpolation method to use in the vertical direction when re-sampling 4:4:4 data to or from 4:2:0.

Gamma Corrector

Table 4–13 shows the Gamma Corrector MegaCore function parameters. These parameters can be set in the MegaWizard interface (see page 2–12).

Table 4–13. Gamma Corrector Parameter Settings

Parameter	Value	Description
Image width	32–1920 Default = **640**	Choose the required image width in pixels.
Image height	32–1080 Default = **480**	Choose the required image height in pixels.
Bits per pixel per color plane	4–16 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'.

Table 4–14 on page 4–34 describes control register map for the Gamma Corrector. These registers are accessed via an Avalon-MM Slave port as described in "Avalon-MM Slave Interfaces" on page 3–13, and must be set by external hardware.

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 3–13.

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 4–14. Gamma Corrector 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 address to 0 will cause the Gamma Corrector MegaCore function to stop the next time control information is read. Refer to "Avalon-MM Slave Interfaces" on page 3–13 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 3–13 for full details.
$2–2^N+1$ 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.

2D FIR Filter

Table 4–15 and Table 4–16 show the 2D FIR Filter MegaCore function parameters. These parameters can be set in the MegaWizard interface (see page 2–13).

Table 4–15. 2D FIR Filter Parameter Settings: General (Part 1 of 2)

Parameter	Value	Description
Image width	32–1920 Default = **640**	Choose the required image width in pixels.
Image height	32–1080 Default = **480**	Choose the required image height in pixels.

Table 4–15. 2D FIR Filter Parameter Settings: General (Part 2 of 2)

Parameter	Value	Description
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'.
Input: Bits per pixel per color plane	4–20 Default = **8**	Choose the number of bits per pixel (per color plane).
Input: Data type	**Unsigned**, Signed	Choose whether input is unsigned or signed 2's complement.
Input: Guard bands	On or **Off**	Turn this option on to enable a defined input range.
Input: Max	1,048,575 to -524,288 Default = **1**	Set input range maximum value. *(1)*
Input: Min	1,048,575 to -524,288 Default = **1**	Set input range minimum value. *(1)*
Output: Data type	**Unsigned**, Signed	Choose whether output is unsigned or signed 2's complement.
Output: Guard bands	On or **Off**	Turn this option on to enable a defined output range.
Output: Max	1,048,575 to -524,288 Default = **1**	Set output range maximum value. *(2)*
Output: Min	1048575 to -524288 Default = **1**	Set output range minimum value. *(2)*
Discard fraction bits by:	**Round values to nearest integer**, Truncate values to integer	Choose the method for discarding fractional bits resulting from the FIR calculation.
Convert from signed to unsigned by:	**Replacing negative values with zero**, Replacing negative values with absolute value, Ignore negative values	Choose the method for signed to unsigned conversion of the FIR results.
Constrain to range by:	**Saturating to min and max values**, Ignore range overflow and underflow,	Choose the method used to constrain the output to a range.

Table 4–16. 2D FIR Filter Parameter Settings: Coefficients (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.
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 4–16. 2D FIR Filter Parameter Settings: Coefficients (Part 2 of 2)

Parameter	Value	Description
Enable symmetric mode	**On** or Off	When turned 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 unset 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 this option on if you want the fixed-point type used to store 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 = **9**	Specifies the number of fractional bits for the fixed point type used to store the coefficients.
Prior to multiply results by:	**1**,2,4,8,16,32,64	Choose the scale factor for the FIR result. The scaling factor can be useful if you require a wider range output on an existing coefficient set.

Notes to Table 4–16

(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 within the specified range of maximum and minimum guard bands values.

2D Median Filter

Table 4–17 shows the 2D Median Filter MegaCore function parameters. These parameters can be set in the MegaWizard interface (see page 2–18).

Table 4–17. 2D Median Filter Parameter Settings (Part 1 of 2)

Parameter	Value	Description
Image width	32–1920 Default = **640**	Choose the required image width in pixels.

Table 4–17. 2D Median Filter Parameter Settings (Part 2 of 2)

Parameter	Value	Description
Image height	32–1080 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 4–18 shows the Alpha Blending Mixer MegaCore function parameters. These parameters can be set in the MegaWizard interface (see page 2–19).

Table 4–18. Alpha Blending Mixer 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).
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'.
Number of layers being mixed	**2**–8	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 this option is turned 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 turned off, no alpha data sink ports are generated, and the image layers are fully opaque.
Alpha bits per pixel per color plane	**2**, 4, 8	Choose the number of bits used to represent the alpha coefficient.
Width	32–1920 Default = **640** or **320**	Choose the required image width in pixels for each image layer (background, layer 2, layer 3, layer 4, layer 5, layer 6, layer 7, foreground). No layer width can be greater than the background layer width. The default background layer width is 640; all other layer widths default to 320.

Table 4–18. Alpha Blending Mixer Parameter Settings (Part 2 of 2)

Parameter	Value	Description
Height	32–1080 Default = **480** or **240**	Choose the required image height in pixels for each image layer (background, layer 2, layer 3, layer 4, layer 5, layer 6, layer 7, foreground). No layer height can be greater than the background layer height. The default background layer height is 480; all other layer heights default to 240.

Table 4–19 on page 4–38 describes the Alpha Blending Mixer control register map. These registers are accessed via an Avalon-MM Slave port as described in "Avalon-MM Slave Interfaces" on page 3–13 and must be set by external hardware. 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 4–19. Alpha Blending Mixer 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 address to 0 will cause the Alpha Blending Mixer MegaCore function to stop the next time control information is read. Refer to "Avalon-MM Slave Interfaces" on page 3–13 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 3–13 for full details.
2–4		Unused
5	Layer 1 X	Offset in pixels from the left edge of the background to the left edge of layer 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.
6	Layer 1 Y	Offset in pixels from the top edge of the background to the top edge of layer 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.
7	Layer 1 Active	Layer 1 is only displayed if the zeroth bit of this control register is set. 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.
8	Layer 2 X	….
….	….	….

Note to Table 4–19:
(1) The rows in the table are repeated in ascending order for each layer from 1 to the foreground layer.

Scaler

Table 4–20, Table 4–21, and Table 4–22 show the Scaler MegaCore function parameters. These parameters can be set in the MegaWizard interface (see page 2–20).

Table 4–20. Scaler Parameter Settings: Resolution

Parameter	Value	Description
Run-time control of image size and clipping	On or **Off**	Turn on to enable run-time control of image size and clipping. The input and output size parameters control the maximum values when this option is selected.
Input image width	32–1920, Default = **1024**	Choose the required input width in pixels.
Input image height	32–1080, Default = **768**	Choose the required input height in pixels.
Output image width	32–1920, Default = **640**	Choose the required output width in pixels.
Output image height	32–920, 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.
Enable image clipping	On or **Off**	When this option is turned on, the input image is clipped. If run-time control is also enabled, the clipping rectangle is specified using the Avalon-MM interface. If run-time control is disabled, the clipping rectangle is specified by the Width, X offset, Height, and Y offset controls.
Width	32 to input image width, Default = **1024**	Specify the width of the clipping rectangle for the input image.
X offset	positive integer, Default = **0**	Specify the x coordinate for the left edge of the clipping rectangle. 0 is the left edge of the input image. *Note (1)*
Height	32 to input image height, Default = **768**	Specify the height of the clipping rectangle for the input image.
Y offset	positive integer, Default = **0**	Specify the y coordinate for the top edge of the clipping rectangle. 0 is the top edge of the input image. *Note (2)*

Notes to Table 4–20:
(1) The X offset value plus the clipping width must be less than or equal to the input image width.
(2) The Y offset value plus the clipping height must be less than or equal to the input image height.

Table 4–21. Scaler Parameter Settings: Algorithm and Precision

Parameter	Value	Description
Scaling Algorithm	Nearest Neighbor, Bilinear, Bicubic, **Polyphase**	Choose the scaling algorithm. See pages 4–14 to 4–16 for more information about these options.
Number of vertical taps	3–16, Default = **4**	Specify the number of vertical taps.
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 this option on if you want the fixed-point type used to store 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 this option on if you want the fixed-point type used to store 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 4–22. Scaler Parameter Settings: Coefficients (Part 1 of 2)

Parameter	Value	Description
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.

Table 4–22. Scaler Parameter Settings: Coefficients (Part 2 of 2)

Parameter	Value	Description
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.
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.

Table 4–23 describes the Scaler control register map. These registers are accessed via an Avalon-MM Slave port as described in "Avalon-MM Slave Interfaces" on page 3–13 and must be set by external hardware. 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 4–23. Scaler Control Register Map (Part 1 of 2)

Address	Register	Description
0	Go	The zeroth bit of this register is the `Go` bit, all other bits are unused. Setting this address to 0 will cause the Scaler MegaCore function to stop the next time control information is read. Refer to "Avalon-MM Slave Interfaces" on page 3–13 for full details.
1	Status	The Scaler MegaCore function sets this address to 0 between frames. It is set to 1 while the core is processing data and cannot be stopped.
2	Input width	The width of the input frames in pixels.
3	Input height	The height of the input frames in pixels.
4	Output width	The width of the output frames in pixels.
5	Output height	The height of the output frames in pixels.
6	Clipping x offset	The x coordinate for the left edge of the clipping rectangle. 0 is the left edge of the input image. If clipping is disabled, this address is ignored
7	Clipping y offset	The y coordinate for the top edge of the clipping rectangle. 0 is the top edge of the input image. If clipping is disabled, this address is ignored.
8	Clipping width	The height of the clipping rectangle for the input image.
9	Clipping height	The width of the clipping rectangle for the input image.
10 to $9+N_V \times P_V$	Vertical coefficient data	If runtime loading of coefficients is enabled with N_V vertical taps and P_V vertical phases, these addresses are for the vertical coefficients. They are laid out with the taps in order for each phase beginning with tap 0, phase 0 at address 10; tap 1, phase 0 at address 11 and continuing up to tap N_V–1, phase P_V–1 at address $9+N_V \times P_V$.

Table 4–23. Scaler Control Register Map (Part 2 of 2)

Address	Register	Description
$10+N_v \times P_v$ to $9+N_v \times P_v+N_h \times P_h$	Horizontal coefficient data	The horizontal coefficients are arranged in the same way as the vertical coefficients and lie immediately after them in the address space.

Deinterlacer

Table 4–24 shows the Deinterlacer MegaCore function parameters. These parameters can be set in the MegaWizard interface (see page 2–25).

Table 4–24. Deinterlacer Parameter Settings

Parameter	Value	Description
Image width	32–1920 Default = **640**	Choose the required image width in pixels.
Image height	32–1080 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, Default = **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'.
Deinterlacing Method	**Bob - Scanline Duplication**, Bob - Scanline Interpolation, Weave	See "Deinterlacing Methods" on page 4–23.
Base address of frame buffers	Any 32-bit value	Address of the frame buffers in external memory when the `Weave` deinterlacing method is used.

Line Buffer Compiler

Table 4–25 shows the Line Buffer Compiler parameters. These parameters can be set in the MegaWizard interface (see page 2–27).

Table 4–25. Line Buffer Compiler Parameter Settings

Parameter	Value	Description
Line length	1–1920, Default = **64**	The length of each line buffer in bits.
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.

Note to Table 4–25:
(1) The width of the output port is equal to the line width multiplied by the number of lines and must not exceed 64.

Signals

Tables 4–26 to 4–34 list the input and output signals for the Video and Image Processing Suite MegaCore functions.

Color Space Converter

Table 4–26 shows the input and output signals for the Color Space Converter MegaCore function.

Table 4–26. Color Space Converter 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 of input port din. Pixel data is transferred into the MegaCore function over this bus.
din_ready	Out	Avalon-ST ready signal of input port din. This signal indicates when the MegaCore function is ready to receive data.
din_valid	In	Avalon-ST valid signal of input port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus of output port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_ready	In	Avalon-ST ready signal of output port dout. This signal indicates when the MegaCore can legally output data.
dout_valid	Out	Avalon-ST valid signal of output port dout. This signal is asserted when the MegaCore function is outputting data.

Chroma Resampler

Table 4–27 shows the input and output signals for the Chroma Resampler MegaCore function.

Table 4–27. Chroma Resampler 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 4–27. Chroma Resampler Signals (Part 2 of 2)

Signal	Direction	Description
din_data	In	Avalon-ST data bus of input port din. Pixel data is transferred into the MegaCore function over this bus.
din_ready	Out	Avalon-ST ready signal of input port din. This signal indicates when the MegaCore function is ready to receive data.
din_valid	In	Avalon-ST valid signal of input port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus of output port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_ready	In	Avalon-ST ready signal of output port dout. This signal indicates when the MegaCore can legally output data.
dout_valid	Out	Avalon-ST valid signal of output port dout. This signal is asserted when the MegaCore function is outputting data.

Gamma Corrector

Table 4–28 shows the input and output signals for the Gamma Corrector MegaCore function.

Table 4–28. Gamma Corrector 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 of input port din. Pixel data is transferred into the MegaCore function over this bus.
din_ready	Out	Avalon-ST ready signal of input port din. This signal indicates when the MegaCore function is ready to receive data.
din_valid	In	Avalon-ST valid signal of input port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus of output port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_ready	In	Avalon-ST ready signal of output port dout. This signal indicates when the MegaCore can legally output data.
dout_valid	Out	Avalon-ST valid signal of output port dout. This signal is asserted when the MegaCore function is outputting data.

Table 4–28. Gamma Corrector Signals (Part 2 of 2)

Signal	Direction	Description
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.
gamma_lut_test_writeack	In	Test port associated with Avalon-MM slave port gamma_lut. This port exists for internal testing purposes only and should not be connected in user designs.
gamma_lut_test_writetog	Out	Test port associated with Avalon-MM slave port gamma_lut. This port exists for internal testing purposes only and should not be connected in user designs.

2D FIR Filter

Table 4–29 shows the input and output signals for the 2D FIR Filter MegaCore function.

Table 4–29. 2D FIR 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 of input port din. Pixel data is transferred into the MegaCore function over this bus.
din_ready	Out	Avalon-ST ready signal of input port din. This signal indicates when the MegaCore function is ready to receive data.
din_valid	In	Avalon-ST valid signal of input port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus of output port dout. Pixel data is transferred out of the MegaCore function over this bus.

Table 4–29. 2D FIR Filter Signals (Part 2 of 2)

Signal	Direction	Description
dout_ready	In	Avalon-ST ready signal of output port dout. This signal indicates when the MegaCore can legally output data.
dout_valid	Out	Avalon-ST valid signal of output port dout. This signal is asserted when the MegaCore function is outputting data.

2D Median Filter

Table 4–30 shows the input and output signals for the 2D Median Filter MegaCore function.

Table 4–30. 2D Median 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 of input port din. Pixel data is transferred into the MegaCore function over this bus.
din_ready	Out	Avalon-ST ready signal of input port din. This signal indicates when the MegaCore function is ready to receive data.
din_valid	In	Avalon-ST valid signal of input port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus of output port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_ready	In	Avalon-ST ready signal of output port dout. This signal indicates when the MegaCore can legally output data.
dout_valid	Out	Avalon-ST valid signal of output port dout. This signal is asserted when the MegaCore function is outputting data.

Alpha Blending Mixer

Table 4–31 shows the input and output signals for the Alpha Blending Mixer MegaCore function.

Table 4–31. 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 *(1)*	In	Avalon-ST alpha data bus of input port din for layer *N*. Pixel data is transferred into the MegaCore function over this bus.
alpha_in_N_ready *(1)*	Out	Avalon-ST alpha ready signal of input port din for layer *N*. This signal indicates when the MegaCore function is ready to receive data.
alpha_in_N_valid *(1)*	In	Avalon-ST alpha valid signal of input port din for layer *N*. This signal identifies the cycles when the port should input data.
din_N_data	In	Avalon-ST data bus of input port din for layer *N*. Pixel data is transferred into the MegaCore function over this bus.
din_N_ready	Out	Avalon-ST ready signal of input port din for layer *N*. This signal indicates when the MegaCore function is ready to receive data.
din_N_valid	In	Avalon-ST valid signal of input port din for layer *N*. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus of output port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_ready	In	Avalon-ST ready signal of output port dout. This signal indicates when the MegaCore can legally output data.
dout_valid	Out	Avalon-ST valid signal of output port dout. This signal is asserted when the MegaCore function is outputting 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.

Table 4–31. Alpha Blending Mixer Signals (Part 2 of 2)

Signal	Direction	Description
control_av_writedata	In	Avalon-MM writedata bus of slave port mix_control. These input lines are used for write transfers.
control_test_writeack	In	Test port associated with Avalon-MM slave port mix_control. This port exists for internal testing purposes only and should not be connected in user designs.
control_test_writetog	Out	Test port associated with Avalon-MM slave port mix_control. This port exists for internal testing purposes only and should not be connected in user designs.

Note to Table 4–31

(1) These ports are only present if alpha blending is enabled. 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 4–32 shows the input and output signals for the Scaler MegaCore function.

Table 4–32. 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 of input port din. Pixel data is transferred into the MegaCore function over this bus.
din_ready	Out	Avalon-ST ready signal of input port din. This signal indicates when the MegaCore function is ready to receive data.
din_valid	In	Avalon-ST valid signal of input port din. This signal identifies the cycles when the port should input data.
dout_data	Out	Avalon-ST data bus of output port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_ready	In	Avalon-ST ready signal of output port dout. This signal indicates when the MegaCore can legally output data.
dout_valid	Out	Avalon-ST valid signal of output port dout. This signal is asserted when the MegaCore function is outputting data.
control_av_address *(1)*	In	Avalon-MM address bus of slave port mix_control. Specifies a word offset into the slave address space.

Table 4–32. Scaler Signals (Part 2 of 2)

Signal	Direction	Description
control_av_chipselect *(1)*	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 *(1)*	Out	Avalon-MM readdata bus of slave port mix_control. These output lines are used for read transfers.
control_av_write *(1)*	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 *(1)*	In	Avalon-MM writedata bus of slave port mix_control. These input lines are used for write transfers.
control_test_writeack *(1)*	In	Test port associated with Avalon-MM slave port mix_control. This port exists for internal testing purposes only and should not be connected in user designs.
control_test_writetog *(1)*	Out	Test port associated with Avalon-MM slave port mix_control. This port exists for internal testing purposes only and should not be connected in user designs.

Note to Table 4–31
(1) These ports are only present if run-time coefficient control or run-time resolution control is enabled.

Deinterlacer

Table 4–33 shows the input and output signals for the Deinterlacer MegaCore function.

Table 4–33. Deinterlacer 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 of input port din. Pixel data is transferred into the MegaCore function over this bus.
din_ready	Out	Avalon-ST ready signal of input port din. This signal indicates when the MegaCore function is ready to receive data.
din_valid	In	Avalon-ST valid signal of input port din. This signal identifies the cycles when the port should input data.

Table 4–33. Deinterlacer Signals (Part 2 of 2)

Signal	Direction	Description
dout_data	Out	Avalon-ST data bus of output port dout. Pixel data is transferred out of the MegaCore function over this bus.
dout_ready	In	Avalon-ST ready signal of output port dout. This signal indicates when the MegaCore can legally output data.
dout_valid	Out	Avalon-ST valid signal of output port dout. This signal is asserted when the MegaCore function is outputting data.
read_master_av_address	Out	Avalon-MM address bus of read_master master port. Specifies a byte address in the Avalon-MM address space.
read_master_av_read	Out	Avalon-MM read signal of read_master 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 master port. These input lines carry data for read transfers.
read_master_av_readdatavalid	In	Avalon-MM readdatavalid signal of read_master master port. This signal is asserted by the system interconnect fabric when requested read data has arrived.
read_master_av_waitrequest	In	Avalon-MM waitrequest signal of read_master master port. Asserted by the system interconnect fabric to cause the master port to wait.
write_master_av_address	Out	Avalon-MM address bus of write_master master port. Specifies a byte address in the Avalon-MM address space.
write_master_av_write	Out	Avalon-MM write signal of write_master 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 master port. These output lines carry data for write transfers.
write_master_av_waitrequest	In	Avalon-MM waitrequest signal of write_master master port. Asserted by the system interconnect fabric to cause the master port to wait.

Note to Table 4–33:
(1) The write_master_* and read_master_* signals are only present when you are using Weave deinterlacing.

Line Buffer Compiler

Table 4–34 shows the input and output signals for the Line Buffer Compiler MegaCore function.

Table 4–34. 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.

MegaCore Verification

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

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

References

1. *E Catmull and R Rom. A class of local interpolating splines. Computer Aided Geometric Design, pages 317–326, 1974.*

In addition, Altera application notes, white papers, and user guides providing more detailed explanations of how to effectively design with MegaCore functions and the Quartus II software are available at the Altera website: www.altera.com.