LogiCORE IP Video Deinterlacer v1.0

Product Guide

PG017 October 19, 2011

XILINX®

Table of Contents

LogiCORE IP Video Deinterlacer v1.0

Introduction

The Xilinx Video Deinterlacer LogiCORE™ IP provides a flexible video processing block for deinterlacing video into a progressive video structure. The core supports image sizes up to 2kx2k with YUV 4:4:4, 4:2:2 or 4:20 and RGB image formats. The core is programmable through a comprehensive register interface for setting and controlling internal operations and more using logic or a microprocessor. An interrupt status mechanism is used for smooth transitioning of changing input video streams to alternative raster structures and planes. The LogiCore IP is provided with two different interfaces: General Purpose Processor and EDK pCore AXI-4 Lite.

Features

- Supports video frame sizes up to 2048x2048 pixels
- Supports video frames sizes down to 128x128
- Supports YUV-4:4:4, 4:2:0 and 4:2:0 and RGB color spaces
- Supports 8, 10 or 12-bit color depth per plane
- Provides smooth transition of output video when changing video standards
- Progressive Segmented Frame (PsF) conversion
- Progressive or Interlaced Format Pass Through
- AXI-MM interface or 3 Port MPMC interface for highest quality deinterlacing
- Provides processor interfaces for EDK pCore and General Purpose Processor
- Supports easy integration with other Xilinx Video IP Cores, including the OSD, VDMA and Video Scaler

LogiCORE IP Facts Table					
Core Specifics					
Supported Device Family (1)	Spartan®-6, Virtex®-6				
Supported User Interfaces	AXI4, AXI4-Lite, AXI4-Stream, General Purpose Processor (GPP)				
	Resources				**Frequency**
Configuration	**LUTs**	**FFs**	**DSP Slices**	**Block RAMs**	**Max. Freq.**
Spartan-6	See Table 1-1				150 MHz
Virtex-6	See Table 1-2				225 MHz
Provided with Core					
Documentation	Product Specification, Data Sheet, User Guide				
Design Files	Netlist, EDK pCore files				
Example Design	Not Provided				
Test Bench	VHDL (2)				
Constraints File	Not Provided				
Simulation Model	VHDL, Verilog Structural, C Model (2)				
Tested Design Tools					
Design Entry Tools	Integrated Software Environment (ISE) 13.3 Xilinx Platform Studio (XPS) 13.3				
Simulation (3)	Mentor Graphics ModelSim				
Synthesis Tools	Xilinx Synthesis Technology (XST)				
Support					
Provided by Xilinx @ www.xilinx.com/support					

1. For a complete listing of supported devices, see the release notes for this core.
2. HDL test bench and C Model available on the product page on Xilinx.com at http://www.xilinx.com/products/ipcenter/EF-DI-DEINTERLACER.htm
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

XILINX®

Chapter 1

Overview

A vast majority of display technologies and video compression techniques use progressive scanning techniques for applications. These technologies require a way to convert interlaced material to progressive scanning methods. The Xilinx Video Deinterlacer core provides the mechanism for achieving this goal.

The Xilinx Video Deinterlacer converts live incoming interlaced video streams into progressive video streams. This process is performed in real time as the input video passes through the video deinterlacer.

By definition, interlaced images have temporal motion between the two fields that comprise an interlaced frame. The conversion to a progressive format recombines these two fields into one single frame. The raw recombination of interlaced video streams results in unsightly motion artifacts in the progressive output image. For this reason, the video deinterlacer uses additional motion tracking and diagonal edge enhancement techniques to ensure that these artifacts are removed where possible. This results in a high-quality progressive output image.

In addition to deinterlacing, the video deinterlacer fully supports both progressive pass through, "Progressive Segmented Frames" (PsF) and "Pull down" encoded streams.

The core supports a wide range of industry standard video encoding and packing methods, including:

- 8, 10 or 12 bits per pixel
- YUV or RGB color spaces (static or dynamically configurable)
- 4:2:0, 4:2:2 or 4:4:4 packing (static or dynamically configurable)

The video deinterlacer requires an external memory store to maintain a three field triple buffer. The core interfaces to external memory using a Xilinx VFBC protocol port or axi-interconnect through the AXI-MM port.

The video deinterlacer supports highly scalable resolutions with a range of 128x128 up to 2048x2048, such as:

- Supported standard SD formats are 480i, 480p, 576i, 576p
- Supported standard HD formats are 720p, 1080i, 1080p
- Digital Cinema 2K
- All PC resolutions (for example, 640x480, 1024x768, 1280x1024, 1920x1200)

The core is highly configurable and can be optimized for the smallest FPGA footprint.

Figure 1-1 illustrates the internal architecture of the video deinterlacer. The video deinterlacer comprises two main video processing kernels and a memory controller interface.

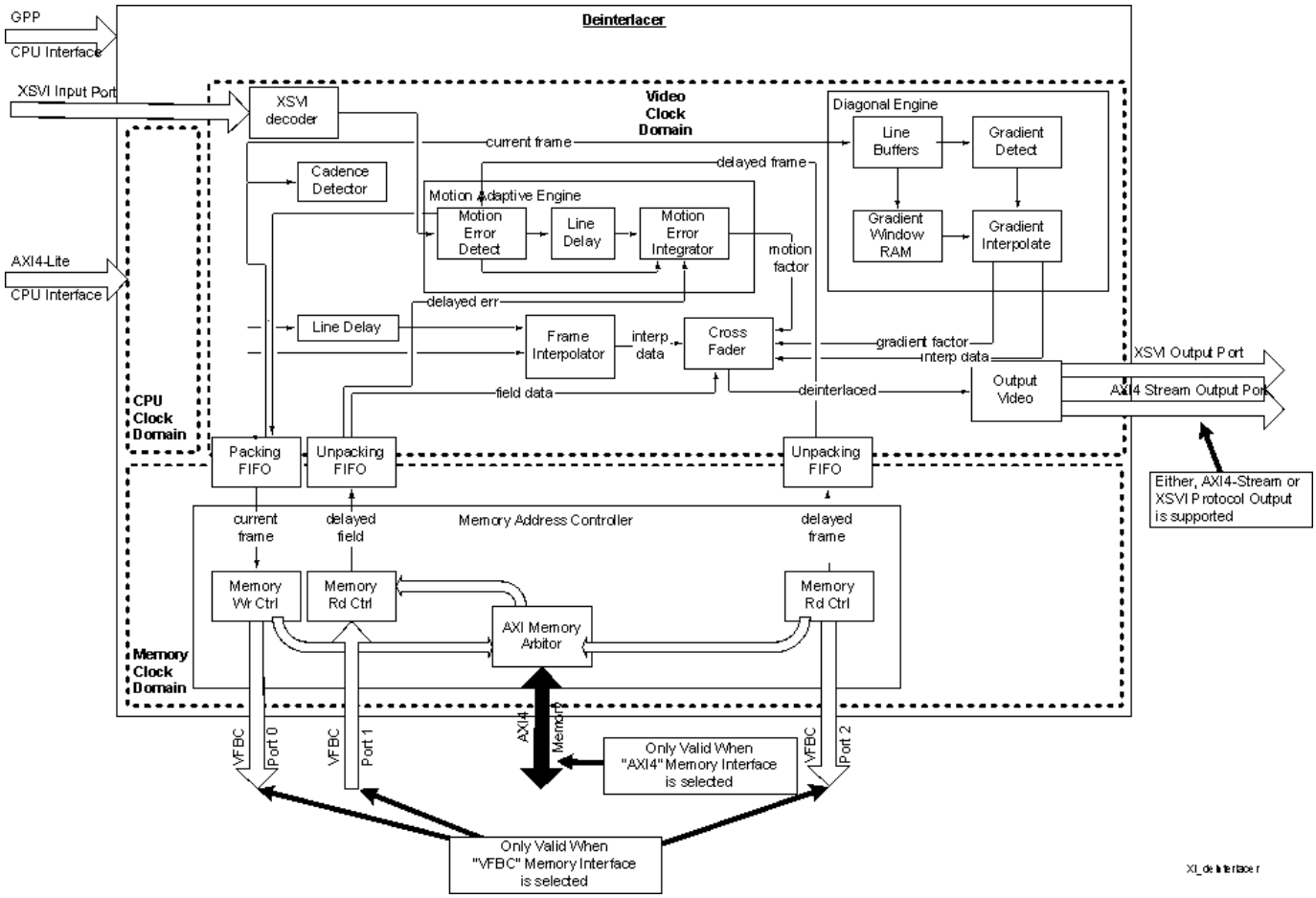

Figure 1-1: **Architecture of Video Deinterlacer**

The deinterlacer is a stream-based core that processes interlaced video on the fly to produce a progressive video output. In a multiple video standard environment, the deinterlacer is software programmable to process interlaced, progressive or Progressive Segmented Frame (PsF) video structures, allowing the video deinterlacer to remain in the system datapath at all times.

The deinterlacer is fully autonomous in its processing, but the deinterlacing effects of the kernels can be altered by system software on a dynamic basis.

The deinterlacing algorithm is based on a combination of motion adaptive concepts combined with diagonal interpolation techniques, resulting in a high quality deinterlaced image.

Figure 1-2 shows a traditional output from a motion adaptive deinterlacer. The staircase effect of fast moving video causes a field interpolation distortion effect on the output video.

Figure 1-2: **Classic Motion Adaptive Deinterlacing Techniques**

Using the deinterlacer core, a blend of motion and diagonal algorithms are combined to create the image in Figure 1-3. The deinterlacer's algorithms recognize motion and detect diagonal vectors. These are combined to form a cleaner pixel that is used in the output video.

Figure 1-3: **Xilinx Video Deinterlacer Deinterlacing Algorithm**

Standards Compliance

The Video Deinterlacer core is compliant with the AXI4-Lite and AXI4-Stream standards as defined in the AXI Reference Guide (UG761).

Feature Summary

Applications include:

- Conversion of interlaced SD TV to progressive SD
- Conversion of CCD image data to a progressive image
- Reconstruction of original 24P film rate from an interlaced source
- Combined with Xilinx Video Scaler, SD to HD up-conversion system

Licensing

Ordering Information

The Video Deinterlacer core is provided under the SignOnce IP Site License and can be generated using the Xilinx CORE Generator system. The CORE Generator system is shipped with Xilinx ISE Design Suite software.

A simulation evaluation license for the core is shipped with the CORE Generator system. To access the full functionality of the core, including FPGA bitstream generation, a full license must be obtained from Xilinx. For more information, visit the Video Deinterlacer product page.

Contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx IP Center.

Licensing Options

The Xilinx video deinterlacer LogiCORE system provides three licensing options. After installing the required Xilinx ISE software and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess the core functionality with your own design and demonstrates the various interfaces on the core in simulation. (Functional simulation is supported by a dynamically-generated HDL structural model.)

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place and route the design, evaluate timing, and perform back-annotated gate-level simulation of the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

Full

The Full license key is provided when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Back annotated gate-level simulation support
- Full implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time outs

Simulation License

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Performance

Deinterlacing Quality Configurations

The deinterlacer comprises these possible quality levels of deinterlacing:

- On the fly field interpolation (lowest quality)
- On the fly field interpolation with diagonal enhancement
- Motion adaptive
- Motion adaptive and diagonal enhancement (highest quality)

The deinterlacer can either be statically configured at core generation time or dynamically configured via the AXI4-Lite interface to perform any of the previous deinterlacing techniques on input video.

Inclusion of the motion adaptive (C_MOTION=1) core requires a MPMC or AXI-MM based external memory interface. The external infterface is used to provide the highest possible quality of deinterlacing. Opting out of the motion adaption core (C_MOTION=0) removes the need for an external memory interface and significantly reduces the FPGA resources required. However, the trade-off is lower quality of the output image. The VFBC/AXI-MM interface ports are not used in this configuration.

Inclusion of the diagonal (C_DIAG=1) core requires only standard FPGA resources (DSP and block RAM) with the benefit of increased image quality.

Latency

Latency equals the average approximate 3 video lines from first pixel entering the core to first pixel coming out of video output port.

Throughput

The deinterlacer creates 2 pixels for every input pixel. Due to this, the deinterlacer requires that the video clock be at minimum twice the video input pixel rate, to allow the internal processing enough clock cycles to generate the output pixels.

There is a 1 line push back buffer at the input of the deinterlacer, to allow for a small amount of sporadic pixel loading into the deinterlacer. But systems that may exhibit more fluctations on input data loading should consider external line buffer blocks that are beyond the scope of the deinterlacer.

There is a 1000 pixel output push back buffer, to allow for small fluctations in the ability for a downstream component to accept data.

If either the input or output buffers overflow, the deinterlacer will raise an interrupt and automatically flush the video pipe and attempt to resynchronise with the passing video on the next frame boundary. All input video will be dropped during this resynchronisation phase.

Resource Utilization

Following are typical clock frequencies for the target families:

- Spartan-6: 150 MHz
- Virtex-6: 225 MHz

The maximum achievable clock can vary and can depend on the size of the device, various aspects of the system design, and other variables.

Resources required for Spartan-6 and Virtex-6 are shown in the following tables.

Table 1-1: **Virtex-6 Resource Estimates**

Feature	Quality	Memory Interface	BRAM 36bit	FIFO	FF	LUT	DSP48E1
Basic Field Interpolation,	Low	none	2	4 ~ 6	842 ~ 997	895 ~ 1010	12
Basic Field Interpolation with diagonal enhancement	Average	none	4	4 ~ 6	1810 ~ 2198	2008 ~ 2527	25
Motion based, no diagonal, 32-bit AXI-MM	High	AXI 32-bit	10	10 ~ 12	2890 - 3084	3129 ~ 3201	14
Full Motion & Diagonal, 32-Bit AXI-MM	Highest	AXI 32-bit	10	10 ~ 12	3821 ~ 4237	4136 ~ 4686	27
Motion based, no diagonal, 64-bit VFBC	Highest	VFBC 64-bit	6	16 ~ 18	2438 ~ 2956	2876 ~ 3021	27
Full Motion & Diagonal, 64-bit VFBC	Highest	VFBC 64-bit	6	16 ~ 18	3597 ~ 4013	3975 ~ 4457	27
Extra Features and Incremental Resource Changes							
Add Cadence Processing		-	+1	-	+600	+630	+1
Decrease max video supported width between 128 ~ 1024 pixels	-	-	-1	-2	-		
Increase AXI to 64-bit instead of 32-bit	Highest	AXI 64	+3	-	+40	+50	-
Increase AXI to 128-bit instead of 32-bit	Highest	AXI 128	+9	-	+120	+100	-
Increase AXI to 256-bit instead of 32-bit	Highest	AXI 256	+21	-	+240	+200	-
GPP Mode instead of AXI-Lite slave CPU Interface	-	-	-	-	-200	-360	-

Table 1-2: **Spartan-6 Resource Estimates**

Feature	Quality	Memory Interface	BRAM 18bit	FIFO	FF	LUT	DSP48A1
Basic Field Interpolation	Lowest	none	11 ~ 14	-	1132 ~ 1339	1135 ~ 1248	12
Basic Field Interpolation with diagonal enhancement	Low	none	14 ~ 17	-	2112 ~ 2541	2289 ~ 2811	25
Motion based, no diagonal, 32-bit AXI-MM	High	AXI 32-bit	31 ~ 36	-	4059 ~ 4367	3966 ~ 4027	14

Table 1-2: **Spartan-6 Resource Estimates**

Feature	Quality	Memory Interface	BRAM 18bit	FIFO	FF	LUT	DSP48A1
Full Motion & Diagonal, 32-Bit AXI-MM	Highest	AXI 32-bit	34 ~ 36	-	4901 ~ 5430	4988 ~ 5394	27
Motion based, no diagonal, 64-bit VFBC	Highest	VFBC 64-bit	33 ~35	-	4323 ~ 4912	4462 ~ 4854	27
Full Motion & Diagonal, 64-bit VFBC	Highest	VFBC 64-bit	33 ~ 35	-	5265 ~ 5867	5303 ~ 5776	27
Add Cadence Processing		-	+1	-	+600	+630	+1
Decrease Max Video Supported							
128 ~ 1024 pixels	-	-	-4	-	-		
Increase AXI to 64-bit instead of 32-bit	Highest	AXI 64	+12	-	+120	+140	-
Increase AXI to 128-bit instead of 32-bit	Highest	AXI 128	+19	-	+380	+300	-
Increase AXI to 256-bit instead of 32-bit	Highest	AXI 256	+41	-	+790	+670	-
GPP Mode instead of AXI-Lite slave CPU Interface	-	-	-	-			-

XILINX®

Chapter 2

Core Interfaces and Register Space

This chapter provides detailed descriptions for each interface. In addition, detailed information about configuration and control registers is included.

Port Descriptions

Core Interfaces

Memory Mapped Interface

When configured to support motion based deinterlacing, the Video Deinterlacer requires an external memory port to perform this operation. The core can be configured to support either a single bi-directional AXI4-Memory Mapped interface or a triple-port-VFBC-interface.

The core provides registers to allow you to specify the location in external memory of the data-buffers that are used by the motion tracking algorithm.

Processor Interface

When configured as an EDK pCore an AXI4-Lite interface is made available for use by a system CPU or other AXI master. The processor interfaces gives full access to the Deinterlacer's internal registers and interrupt systems. The internal status of the Deinterlacer can also be monitored through this interface

General Purpose Processor Interface

When configured in General Purpose Processor mode all internal control and status signals are brought to the top level for direct connection to an external controller or tie-offs.

Video Streaming Input Interface

The core has a single video input port. The video input port is always defined to be XSVI protocol, but its width and packing modes are controlled in CoreGen.

Video Streaming Output interface

The core has a single video output port. This port can be configured to be XSVI or AXI4-Streaming Protocol.

Common I/O Signals

The EDK pCore interface and the General Purpose Processor interface share some common global signals. These are :

Port Name	Dir	Width	Description
vid_clk	I	1	Main system video clock. Synchronous to XSVI in and out ports
ce	I	1	Main system video clock enable. Used to throttle data passing through the deinterlacer.
sclr	I	1	asynchronous system reset.
vfbc_clk	I	1	VFBC/AXI master clock. All VFBC or AXI-MM ports are synchronous to this clock
fsync_out	O	1	Frame Synchronization Pulse for down-stream devices such as AXI_VDMA

The cores video interface pins are shown below:

XSVI Input Video Interface			
xsvi_video_data_in	I	[C_STREAMS* C_DEPTH-1:0]	Input video data, packed according to XSVI interface specification.
xsiv_hblank_in	I	1	Input video horizontal blanking, active high
xsvi_vblank_in	I	1	Input video vertical blanking, active high
xsvi_active_video_in	I	1	Input video active video strobe, active high
xsvi_active_chroma_in	I	1	Input video active chroma strobe, active-High. Only used if 422 or 420 packing modes are selected
xsvi_field_id_in	I	1	Input video field id flag
XSVI Output Interface			
xsvi_video_data_out	O	[C_STREAMS* C_DEPTH-1:0]	Output video data, packed according to XSVI interface specification.
xsvi_hblank_out	O	1	Output video horizontal blanking, active-High
xsvi_vblank_out	O	1	Output video vertical blanking, active-High
xsvi_active_chroma_out	O	1	Output video chroma strobe, active high. Only used if 422 or 420 packing modes are selected
xsvi_active_video_out	O	1	Output video active video strobe, active high

xsvi_en_out	O	1	Output video enable strobe
AXI4-Streaming Output Interface			
m_axis_tdata	I	[C_M_AXIS_TDATA_WIDTH-1:0]	Output video data, packed according to AXI4S-XSVI interface specification
m_axis_tkeep	O	[C_M_AXIS_TDATA_WIDTH/8-1:0]	Output video keep strobe
m_axis_tstrb	O	[C_M_AXIS_TDATA_WIDTH/8-1:0]	Output video data strobe
m_axis_tvalid	O	1	Output video data is valid enable
m_axis_tready	I	1	Output video data acknowledge
m_axis_tlast	O	1	Output video end of video line marker

External Memory Interface Signals

When configured as a VFBC Memory interface the following signals are present:

VFBC Port 0			
vfbc0_cmd_reset	O	1	VFBC command reset
vfbc0_cmd_full	I	1	VFBC command full flag
vfbc0_cmd_write	O	1	VFBC command write strobe
vfbc0_cmd_data	O	[31:0]	VFBC command write data
vfbc0_wd_almost_full	I	1	VFBC write data fifo almost full flag
vfbc0_wd_full	I	1	VFBC write data fifo full flag
vfbc0_wd_data	O	[31:0]	VFBC write data
vfbc0_wd_write	O	1	VFBC write data fifo write strobe
vfbc0_wd_reset	O	1	VFBC write data fifo reset
vfbc0_wd_flush	O	1	VFBC write data fifo flush
vfbc0_wd_end_burst	O	1	VFBC write data burst end flag
vfbc0_rd_reset	O	1	VFBC read data fifo reset
vfbc0_rd_read	O	1	VFBC read data read strobe
vfbc0_rd_data	I	[31:0]	VFBC read data
vfbc0_rd_end_burst	O	1	VFBC read data burst end flag
vfbc0_rd_almost_empty	I	1	VFBC read data fifo almost empty flag
vfbc0_rd_empty	I	1	VFBC read data fifo empty flag
VFBC Port 1			

vfbc1_cmd_reset	O	1	VFBC command reset
vfbc1_cmd_full	I	1	VFBC command full flag
vfbc1_cmd_write	O	1	VFBC command write strobe
vfbc1_cmd_data	O	[31:0]	VFBC command write data
vfbc1_wd_almost_full	I	1	VFBC write data fifo almost full flag
vfbc1_wd_full	I	1	VFBC write data fifo full flag
vfbc1_wd_data	O	[31:0]	VFBC write data
vfbc1_wd_write	O	1	VFBC write data fifo write strobe
vfbc1_wd_reset	O	1	VFBC write data fifo reset
vfbc1_wd_flush	O	1	VFBC write data fifo flush
vfbc1_wd_end_burst	O	1	VFBC write data burst end flag
vfbc1_rd_reset	O	1	VFBC read data fifo reset
vfbc1_rd_read	O	1	VFBC read data read strobe
vfbc1_rd_data	I	[31:0]	VFBC read data
vfbc1_rd_end_burst	O	1	VFBC read data burst end flag
vfbc1_rd_almost_empty	I	1	VFBC read data fifo almost empty flag
vfbc1_rd_empty	I	1	VFBC read data fifo empty flag
VFBC Port 2			
vfbc2_cmd_reset	O	1	VFBC command reset
vfbc2_cmd_full	I	1	VFBC command full flag
vfbc2_cmd_write	O	1	VFBC command write strobe
vfbc2_cmd_data	O	[31:0]	VFBC command write data
vfbc2_wd_almost_full	I	1	VFBC write data fifo almost full flag
vfbc2_wd_full	I	1	VFBC write data fifo full flag
vfbc2_wd_data	O	[31:0]	VFBC write data
vfbc2_wd_write	O	1	VFBC write data fifo write strobe
vfbc2_wd_reset	O	1	VFBC write data fifo reset
vfbc2_wd_flush	O	1	VFBC write data fifo flush
vfbc2_wd_end_burst	O	1	VFBC write data burst end flag
vfbc2_rd_reset	O	1	VFBC read data fifo reset
vfbc2_rd_read	O	1	VFBC read data read strobe
vfbc2_rd_data	I	[31:0]	VFBC read data
vfbc2_rd_end_burst	O	1	VFBC read data burst end flag
vfbc2_rd_almost_empty	I	1	VFBC read data fifo almost empty flag
vfbc2_rd_empty	I	1	VFBC read data fifo empty flag

When Configured with a AXI-MM interface the following signals are present:

AXI4-Lite Slave Interface			
m_axi_awaddr	O	[31:0]	AXI Write Address
m_axi_awid	O	[C_M_AXI_THREAD_ID_WIDTH-1]	AXI Write Thread ID
m_axi_awlen	O	[7:0]	AXI Write Burst Length
m_axi_awsize	O	[2:0]	AXI Write Beat Size
m_axi_awburst	O	[1:0]	AXI Write Burst Type
m_axi_awlock	O	1	AXI Write Transaction lock
m_axi_awcache	O	[3:0]	AXI Write Cache Type
m_axi_awprot	O	[2:0]	AXI Write Protection Level
m_axi_awqos	O	[3:0]	AXI Write Quality of Service
m_axi_awvalid	O	1	AXI Write Address Valid
m_axi_awready	I	1	AXI Write Address acknowledge
m_axi_wdata	O	[C_M_AXI_DATA_WIDTH-1:0]	AXI Write Data
m_axi_wstrb	O	[C_M_AXI_DATA_WIDTH/8-1:0]	AXI Write Data Strobes
m_ax_wlast	O	1	AXI Write Burst Last Beat
m_axi_wvalid	O	1	AXI Write Data Valid
m_axi_wready	I	1	AXI Write Data acknowledge
m_axi_bid	I	[C_M_AXI_THREAD_ID_WIDTH-1:0]	AXI Write Response Thread ID
m_axi_bresp	I	2	AXI Write Response
m_axi_bvalid	I	1	AXI Write Response Valid
m_axi_bready	O	1	AXI Write Response Acknowledge
m_axi_arid	O	[C_M_AXI_THREAD_ID_WIDTH-1:0]	AXI Read Thread ID
m_axi_araddr	O	[31:0]	AXI Read Address
m_axi_arlen	O	[7:0]	AXI Read Burst Length
m_axi_arsize	O	[2:0]	AXI Read Burst beat size
m_axi_arburst	O	[1:0]	AXI Read Burst type
m_axi_arlock	O	1	AXI Read Transaction Locked

m_axi_arcache	O	[3:0]	AXI Read Transaction Protection Level
m_axi_arprot	O	[2:0]	AXI Read Cache type
m_axi_arqos	O	[3:0]	AXI Read Quality of Service
m_axi_arvalid	O	1	AXI Read Address Valid
m_axi_arready	I	1	AXI Read Address acknowledge
m_axi_rid	I	[C_M_AXI_THREAD_ID_WIDTH-1:0]	AXI Read Data Thread ID
m_axi_rdata	I	[C_M_AXI_DATA_WIDTH-1:0]	AXI Read Data
m_axi_rresp	I	1	AXI Read Response
m_axi_rlast	I	1	AXI Read Data Burst Last beat strobe.
m_axi_rvalid	I	1	AXI Read Response Valid
m_axi_rready	O	1	AXI Reset Response acknowledge

Configuration Interface Signals

When configured as a general purpose processer interface control mechanism. The following pins are present:

General Purpose Processor Interface			
gpp_update_req	I	1	Internal register update request
gpp_update_done	O	1	Internal register updates are completed
gpp_bypass	I	1	Force deinterlacer pass through
gpp_fs_base_0	I	[31:0]	Base address of Page 0 Field Buffer
gpp_fs_base_1	I	[31:0]	Base address of Page 1 Field Buffer
gpp_fs_base_2	I	[31:0]	Base address of Page 2 Field Buffer
gpp_fs_words	I	[23:0]	Size in 32 bit words of Field Page size
gpp_deint_pack	I	[1:0]	Video Packing format 0 : 4:2:0 Packing 1 : 4:2:2 Packing 2 : 4:4:4 Packing 3 : Reserved
gpp_deint_debug	I	[7:0]	colourisation of output video

gpp_deint_mode	I	[2:0]	deinterlacing algorithm mode 0 = Field interpolation 1 = Motion Adaptive 2 = Diagonal Compensating 3 = Motion & Diagonal Compensating 4 = Line duplication
gpp_deint_thresh_t1	I	[9:0]	Motion adaptive threshold T1
gpp_deint_thresh_t2	I	[9:0]	Motion adaptive threshold T2
gpp_deint_xfade_scale	I	[15:0]	Cross Fade Ratio
gpp_deint_xsize	I	[10:0]	Horizontal frame size
gpp_deint_ysize	I	[10:0]	Vertical input frame size
gpp_deint_col	I	1	Colourspace 0 = YUV 1 = RGB
gpp_deint_black	I	[C_DEPTH *3-1:0]	Definition of black for the active colourspace
gpp_deint_pull_en_22	I	1	pulldown 2:2 controller enable
gpp_deint_pull_en_32	I	1	pulldown 3:2 controller enable
gpp_deint_order	I	1	First Field is odd/even 0 = even (PAL/HD) 1 = odd (NTSC)
gpp_version	O	[31:0]	hardware version id major.minor.rev
gpp_irq_req	O	[15:0]	interrupt request lines

When configured as an EDK pCore the following AXI4-Lite interface is present:

AXI4-Lite Slave Interface			
s_axi_aclk	I	1	CPU clock. The AXI slave interface is synchronous to this clock
s_axi_awaddr	I	[31:0]	AXI Write Address
s_axi_awvalid	I	1	AXI Write Address Valid
s_axi_awready	O	1	AXI Write Address acknowledge
s_axi_wdata	I	[31:0]	AXI Write Data
s_axi_wvalid	I	1	AXI Write Data Valid
s_axi_wready	O	1	AXI Write Data acknowledge
s_axi_bresp	O	2	AXI Write Response
s_axi_bvalid	O	1	AXI Write Response Valid
s_axi_bready	I	1	AXI Write Response Acknowledge

s_axi_araddr	I	[31:0]	AXI Read Address
s_axi_arvalid	I	1	AXI Read Address Valid
s_axi_arready	O	1	AXI Read Address acknowledge
s_axi_rdata	O	[31:0]	AXI Read Data
s_axi_rresp	O	1	AXI Read Response
s_axi_rvalid	O	1	AXI Read Response Valid
s_axi_rready	I	1	AXI Reset Response acknowledge
ip2intc_irpt	O	1	CPU interrupt request. Active High Level interrupt synchronous to s_axi_aclk

Register Space

This section provides the programming interface register information.

All registers power up with 0x0. Only the control, mode and interrupt control registers are reset to 0x0 during a software reset, all other registers retain their current settings.

Table 2-1: **Register Map**

Address	Name	Read/Write	Description
0x0000	control	R/W	General Control register
0x0004	mode	R/W	Deinterlacer modes
0x0008	interrupt control	R/W	Interrupt enable and disable register
0x000C	interrupt status	R/W1C	Interrupt status and clear register
0x0010	height	R/W	Input frame height
0x0014	width	R/W	Input frame width
0x0018	threshold T1	R/W	Motion adaptive threshold T1
0x001C	threshold T2	R/W	Motion adaptive threshold T2
0x0020	cross fade scale	R/W	Cross fade scaling
0x0024	buffer 0 base	R/W	External triple buffer 0 base address
0x0028	buffer 1 base	R/W	External triple buffer 1 base address
0x002C	buffer 2 base	R/W	External triple buffer 2 base address
0x0030	buffer size	R/W	External triple buffer segment size
0x00F0	version	R	Hardware version id
0x0100	soft reset	R/W	Internal soft reset

Table 2-2: **Control Register**

0x0000								Control																				R/W	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																														b	u

Name	Bits	Description
Reserved	31:2	Reserved
Deinterlacer Enable	1	While the deinterlacer is disable, active video passes through the deinterlacer in its original form. ***Note:*** All Blanking information is always stripped by the deinterlacer.
Update Request	0	Setting this bit to '1' arms the deinterlacer to perform a register shadow update on the next frame boundary. Setting this bit to '0' cancels any pending shadow request.

Table 2-3: **Mode Register**

0x0004								Mode																				R/W	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved														d	m	Reserved								p	s	o	pack		c	de	

Name	Bits	Description
Reserved	31:18	Reserved
Colorize Diagonal	17	Enable colorizing output image with diagonal algorithm output
Colorize Motion	16	Enable colorizing output image with motion algorithm output
Reserved	15:9	Reserved
Pull-down Enable 2:2	8	Allow Pull-down detector to automatically control deinterlacer
Pull-down Enable 3:2	7	Allow Pull-down detector to automatically control deinterlacer
PsF Enable	6	Progressive Segmented Frame Enable (PsF mode)
Field Order	5	Sets the first field order for input video When set '1' the field order maps to NTSC / 480i When set '0' the field order maps to PAL / HD / 3G
Packing Format	4:3	Sets the XSVI packing formats used on the input and output When set to 0: 4:2:0 packing is used When set to 1: 4:2:2 packing is used When set to 2: 4:4:4 packing is used

Table 2-3: **Mode Register**

Color Space	2	Colorspace of video When set to '0' YUV colorspace is used When set to '1' RGB colorspace is used
Deinterlacing Algorithm	1:0	Sets the deinterlacing method used When set to '0' pure field interpolating techniques are used When set to '1' only motion adaptive engine is used When set to '2' only the diagonal engine is used When set to '3' both motion and diagonal engines are used

Table 2-4: **Interrupt Control Register**

0x0008								Status																				R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																															

Name	Bits	Description
Reserved	31:12	Reserved
Framestore Rd Err 1	11	Enable Error detection on Third VFBC Port or AXI-MM Port
Framestore Rd Err 0	10	Enable Error detection on Second VFBC Port or AXI-MM Port
Framestore Wr Err	9	Enable Error detection on First VFBC Port or AXI-MM Port
Framestore Wr Marker	8	Enable Framestore integrity checking
Reserved	7	Reserved
Frame Interrupt	6	Enables the video frame border interrupt when set to '1'
Pull-down off	5	Enable pull-down loss detection
Pull-down on	4	Enable pull-down activation detection
Deinterlacer Error	3	Enable internal diagnostic error interrupt
Synch off	2	Enable loss of video lock detector
Synch on	1	Enable lock of input video detector
Update Interrupt	0	Enables the register shadow update done interrupt when set to '1'

Table 2-5: **Interrupt Status Register**

0x000C								Status																				R/W1C			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																															

Name	Bits	Description
Reserved	31:12	Reserved
Framestore Rd Err 1	11	The Third VFBC Port or AXI-MM Port is experiencing FIFO under run

Table 2-5: **Interrupt Status Register**

Framestore Rd Err 0	10	The Second VFBC Port or AXI-MM Port is experiencing FIFO Under run
Framestore Wr Err	9	The First VFBC Port or AXI-MM Port is experiencing FIFO Overrun
Framestore Wr Marker	8	The framestore is experiencing video data frames that do not match the programmed settings
Reserved	7	Reserved
Frame Interrupt	6	A Video frame boundary has passed.
Pull-down off	5	Pull-down detector has seen pull down sequence disappear.
Pull-down on	4	Pull-down detector has found a pull down sequence.
Deinterlacer Error	3	Internal deinterlacer FIFO overrun error.
Synch off	2	Deinterlacer has lost synchronization to input video
Synch on	1	Deinterlacer is synchronized to input video
Update Interrupt	0	A internal register update has occurred

Table 2-6: **Height Register**

0x0010								Height																R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																					Height										

Name	Bits	Description
Reserved	31:11	Reserved
Height	10:0	Input pixel height of video frame

Table 2-7: **Width Register**

0x0014								Width																R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																					Width										

Name	Bits	Description
Reserved	31:11	Reserved
Width	10:0	Input pixel width of video frame

Table 2-8: **Threshold T1 Register**

0x0018								Threshold T1																R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Table 2-8: **Threshold T1 Register**

Reserved	T1

Name	Bits	Description
Reserved	31:10	Reserved
T1 setting	9:0	Motion Adaptive T1 threshold value

Table 2-9: **Threshold T2 Register**

0x001C	Threshold T2	R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																						T2									

Name	Bits	Description
Reserved	31:3	Reserved
T2 setting	0	Motion Adaptive T2 threshold value

Table 2-10: **Cross Fade Scale Register**

0x0020	Cross Fade Scale	R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																xfade															

Name	Bits	Description
Reserved	31:16	Reserved
Cross Fade Scale	15:0	Motion Adaptive cross fade scaling factor. **MUST** be programmed using this equation: scale = (4096*256)/(register T2- register T1)

Table 2-11: **Buffer 0 Register**

0x0024	Buffer 0	R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	Base																														

Name	Bits	Description
Buffer 0 Base	31:0	Base address in external memory of the first field buffer

Table 2-12: **Buffer 1 Register**

0x0028																																R/W
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	Buffer 1
R	Base																															

Name	Bits	Description
Buffer 1 Base	31:0	Base address in external memory of the second field buffer

Table 2-13: **Buffer 2 Register**

0x002C																																R/W
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	Buffer 2
R	Base																															

Name	Bits	Description
Buffer 2 Base	31:0	Base address in external memory of the third field buffer

Table 2-14: **Pull-down High Threshold**

0x0038																																R/W
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	Buffer Size
Reserved								size																								

Name	Bits	Description
Reserved	31:24	Reserved
Pull-down high	23:0	Motion threshold for pull-down high detection

Table 2-15: **Version ID**

0x00F0																																R/W
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	Buffer Size
																				Reserved												

Name	Bits	Description
Major Version	31:28	Major version as a single 4-bit hexadecimal value.
Minor Version	27:20	Minor version as two separate 4-bit hexadecimal values (00 - FF).
Revision Letter	19:16	Revision letter as a hexadecimal character from 'a' - 'f'; mapping is: 0XA->'a', 0xB->'b', 0xC->'c', 0xD->'d', and so on.

Table 2-15: **Version ID**

Patch Revision	15:12	Core Generator Patch Revision.
Reserved	11:0	Reserved

Table 2-16: **Soft Reset**

0x0100								Buffer Size																				R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																															R

Name	Bits	Description
Reserved	31:24	Reserved
Soft Reset	0	Resets the “Mode”, “Control” and “Interrupt Control” registers to zero.

XILINX®

Chapter 3

Customizing and Generating the Core

This chapter includes information on using Xilinx tools to customize and generate the core.

Control Values

In both GPP and pcore modes, the control values are provided dynamically at the input of the deinterlacer and can be changed during run time.

For the pcore version of the core, CORE Generator software provides the GPP core placed in a wrapper, which allows you to parameterize the deinterlacer core in EDK. The ports are driven by registers on a AXI4-Lite bus. The address is decoded in the wrapper. A MicroBlaze™ processor software driver is provided in source code form to drive these ports.

The parameters that can be set dynamically via AXI4-Lite registers are:

- packing: controls the YUV packing mode used; 4:2:2, 4:2:0 or 4:4:4
- kernel mode: controls what deinterlacer algorithms are used
- threshold T1: controls the low motion threshold
- threshold T2: controls the high motion threshold
- cross fade ratio: controls the scaling factor used by the cross fader
- xsize, ysize: controls the active window size of the output video frame
- field order: sets the field order as: HD,PAL or NTSC
- color: selects which color space is processed, YUV or RGB
- black: sets the pixel value for black inside the core, dependent on color space setting
- fswords: set the amount of 32-bit words that are required to store one field of video in the external memory buffer
- fsbase0,1,2: sets the 31-bit base addresses of the three external field buffers
- PsF mode: controls if the deinterlacer is processing interlaced, PsF or progressive image structures
- pull-down mode: controls if the pull-down controller is activated

CORE Generator Tool Graphical User Interface (GUI)

The CORE Generator tool GUI is shown in Figure 3-1. Field descriptions are provided in Parameter Values in the XCO File. Each field sets a parameter used at build time to configure different hardware options.

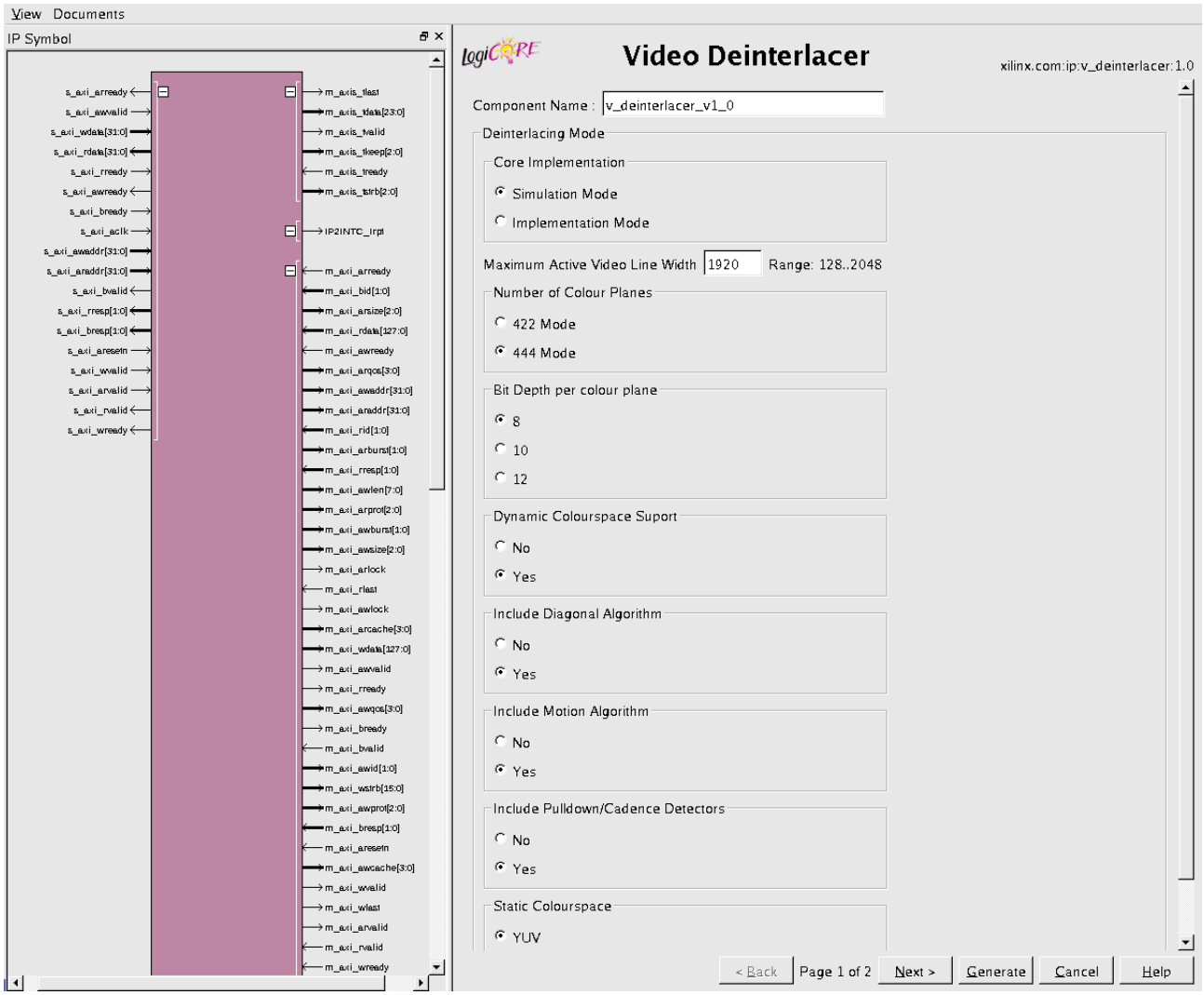

Figure 3-1: **CORE Generator GUI**

EDK pCore GUI

When the deinterlacer core is generated from CORE Generator as an EDK pCore it is generated with each option set to the default value. All customizations of a video deinterlacer pCore are done with the EDK pCore GUI. Figure 3-2 illustrates the EDK pCore GUI for the video deinterlacer. The options in the EDK pCore GUI for the video deinterlacer correspond to the same options in the CORE Generator GUI for the video deinterlacer.

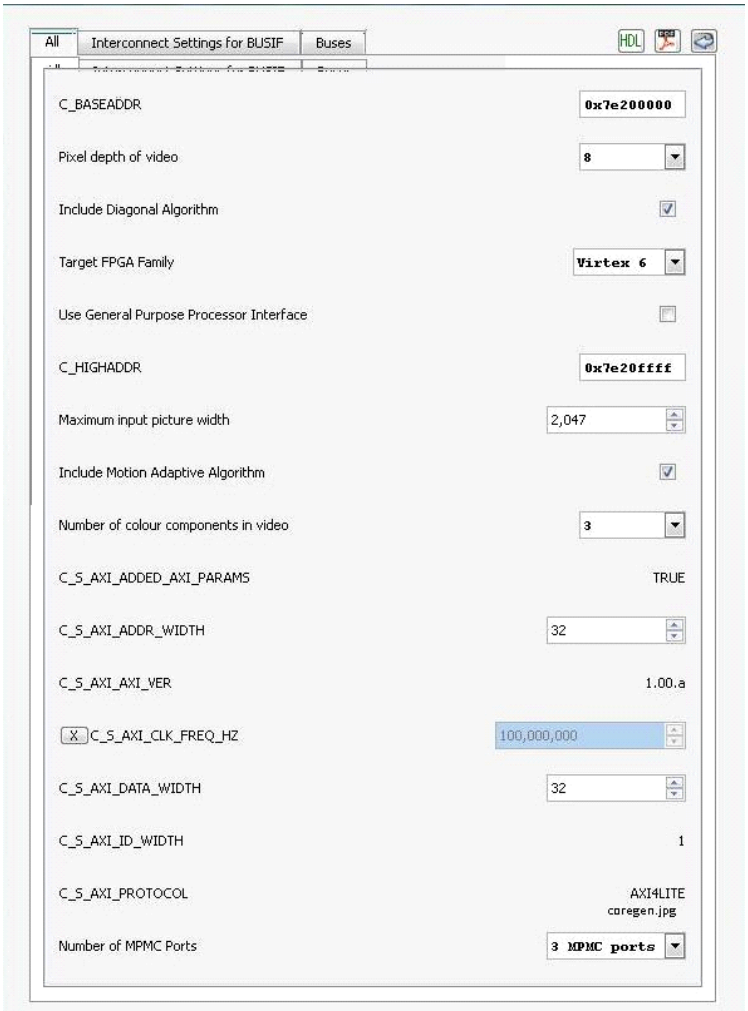

Figure 3-2: **Video Deinterlacer Configuration Screen**

The following table provide the design parameters, allowable values, and descriptions for the video deinterlacer system. Parameter values that are strings or that contain alpha numeric characters must be uppercase.

Table 3-1: **System Parameters**

Parameter Name	Default Value	Allowable Values	Description
C_BASEADDR	0x10000000	Valid Address	System base address
C_HIGHADDR	0x100000FF	Valid Address	System high address
C_FAMILY	Virtex6	Virtex-6 Spartan-6	Target FPGA family
C_MAX_XSIZE	720	128-2048	Maximum raster width supported
C_GPP	0	0,1	Selects between AXI4-Lite or GPP configuration interfaces
C_STREAMS	3	2,3	Number of simultaneous color planes
C_DEPTH	8	8,10,12	Bit depth of a pixel

Table 3-1: **System Parameters**

C_DYN	1	0,1	Dynamic colorspace enabling 0=Static Color Space 1=Dynamic Color Space
C_PULLDOWN	1	0,1	Cadence/Pull-down detection 0=No pull-down detection 1=Full pull-down detection
C_COL	1	0,1	Static color space setting 0=YUV 1= RGB
C_DIAG	1	0,1	Statically include the diagonal kernel
C_MOTION	1	0,1	Statically include the motion kernel
C_OUTPUT_TYPE	1	0,1	Output video interface type 0=XSVI 1=AXI4-Streaming
C_AXI	1	0,1	Memory Interface Protocol 0=VFBC 1=AXI4-MM
C_AXI_TDATA_WIDTH	16	16, 24, 32, 40, 48	Ouput AXI Streaming data width
C_AXI_DATA_WIDTH	64	32, 64, 128, 256	AXI-MM Data Width
C_AXI_THREAD_ID_WIDTH	1	0,1	AXI-MM Thread ID Width

Video Deinterlacer Core Interfaces

There are many video systems that use an integrated MicroBlaze™ processor soft core to dynamically control the parameters within the system. This is especially important when several independent image processing cores are integrated into a single FPGA. The video deinterlacer core can be configured with one of two interfaces: an EDK pCore Interface or a General Purpose Processor Interface.

EDK pCore Interface

The pCore interface creates a core that can be added to an EDK project as a hardware peripheral. This section describes the register set, the pCore driver files, and the I/O signals associated with the video deinterlacer core.

After it is generated by CORE Generate software, the new video deinterlacer pCore is located in the
CORE Generator project directory at *Component_Name*/pcores/deinterlacer_v1_00_a. The pCore
should be copied to the user's *EDK_Project*/pcores directory or to a user pCores repository. The video deinterlacer pCore driver software is located in the CORE Generator project directory at
Component_Name/drivers/deinterlacer_v1_00_a. The driver software should be copied to

the user's
EDK_Project/drivers directory or to a user pCores repository.

pCore Register Set

The pCore interface provides a memory mapped interface for the programmable registers within the core, which are defined in Register Space.

Note: All registers power up with 0x0. Only the control, mode and interrupt control registers are reset to 0x0 during a software reset; all other registers retain their current settings.

Table 3-2: **Video Deinterlacer pCore Memory Mapped Register Set**

Address	Name	Read/Write	Description
0x0000	control	R/W	General Control register
0x0004	mode	R/W	Deinterlacer modes
0x0008	interrupt control	R/W	Interrupt enable and disable register
0x000C	interrupt status	R/W1C	Interrupt status and clear register
0x0010	height	R/W	input frame height
0x0014	width	R/W	input frame width
0x0018	threshold T1	R/W	motion adaptive threshold T1
0x001C	threshold T2	R/W	motion adaptive threshold T2
0x0020	cross fade scale	R/W	cross fade scaling
0x0024	buffer 0 base	R/W	external triple buffer 0 base address
0x0028	buffer 1 base	R/W	external triple buffer 1 base address
0x002C	buffer 2 base	R/W	external triple buffer 2 base address
0x0030	buffer size	R/W	external triple buffer segment size
0x00F0	version	R	hardware version id
0x0100	soft reset	R/W	Internal soft reset

pCore Driver Files

The video deinterlacer pCore includes a C language software driver that the user can use to control the video deinterlacer. A high-level API is provided to hide the details of the video deinterlacer, and application developers are encouraged to use it to access the device features. A low-level API is also provided if developers prefer to access the devices directly through the system registers described in the previous section.

Table 3-3 lists the files that are included with the video deinterlacer pCore driver.

Table 3-3: **Software Driver Files Provided With the Video Deinterlacer pCore**

File Name	Description
xdeint.h	Contains all prototypes of high-level API to access all of the features of the video deinterlacer device.
xdeint.c	Contains the implementation of high-level API to access all of the features of the video deinterlacer device
xdeint_intr.c	Contains the implementation of high-level API to access the interrupt feature of the video deinterlacer device.

Table 3-3: **Software Driver Files Provided With the Video Deinterlacer pCore**

xdeint_sinit.c	Contains static initialization methods for the video deinterlacer device.
xdeint_g.c	Contains a template for a configuration table of video deinterlacer devices. This file is used by the high-level API and is automatically generated to match the video deinterlacer device configuration by EDK/SDK tools when the software project is built.
xdeint_hw.h	Contains low-level API (that is, identifiers and register-level driver API) that can be used to access the video deinterlacer device.
xdeint_i.h	Contains internal functions of the video deinterlacer device driver. The application should never need to invoke any function/macro in this file
example.c	An example that demonstrates how to configure the video deinterlacer device using the high-level API.

Parameter Values in the XCO File

Table 3-4 defines valid entries for the Xilinx CORE Generator (XCO) parameters. Xilinx strongly suggests that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator software GUI to configure the core and perform range and parameter value checking. The XCO parameters are helpful in defining the interface to other Xilinx tools.

Table 3-4: **XCO Parameters**

XCO Parameter	Default	Valid Values
component_name	v_deinterlacer_v1_0_u0	ASCII text using characters: a..z, 0..9 and "_" starting with a letter. Note: "v_deinterlacer_v1_0" is not allowed.
c_col	0	0,1
C_depth	8	8,10,12
C_diag	1	0,1
C_dyn	1	0,1
C_pulldown	1	0,1
C_motion	1	0,1
C_output_type	1	0,1
C_max_size	1920	128-2048
C_streams	3	2,3
C_m_axi_clk_freq_hz	200000000	Positive Integer
C_m_axi_data_width	128	32,64,128,256
C_m_axi_thread_id_width	2	1-4
C_m_axi_tdata_width	24	16,24,32,40,64
C_s_axi_clk_freq_hz	50000000	Positive Integer
C_simulation	1	0,1

Table 3-4: **XCO Parameters**

XCO Parameter	Default	Valid Values
C_gpp	0	0,1
C_axi	1	0,1
C_baseaddr	0x10000000	ASCII text of 32bit hexadecimal value.
C_highaddr	0x100000FF	ASCII text of 32bit hexadecimal value.

Output Generation

The output files generated from the Xilinx CORE Generator software for the Video Deinterlacer core always include EDK pCore specific and CORE Generator specific files. The output files are placed in the project directory.

EDK pCore Files

As part of its output, CORE Generator outputs a set of pCore filesthat can be easily incorporated into an EDK project. The pCore output consists of a hardware pCore and a software driver. The pCore has the following directory structure:

<Component_Name>/edk

- -drivers
  - deinterlacer_v1_01_a
    - data
    - example
    - src
- pcores
  - deinterlacer_v1_00_a
    - data
    - hdl
      - vhdl

File Details

- <project directory>
  This is the top-level directory. It contains xco and other assorted files.

Name	Description
<component_name>.xco	Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_flist.txt	A text file listing all of the output files produced when the customized core was generated in the CORE Generator software.

- <project directory>/<component_name>/edk/pcores/deinterlacer_v1_00_a/data
  This directory contains files that EDK uses to define the interface to the pCore.
- < project directory>/<component_name>/edk/pcores/deinterlacer_v1_00_a/hdl/vhdl

This directory contains the Hardware Description Language (HDL) files that implement the pCore.

- < project directory>/<component_name>/edk/drivers/deinterlacer_v1_01_a /data
  This directory contains files that Software Development Kit (SDK) uses to define the operation of the pCore's software driver.
- < project directory>/<component_name>/edk/drivers/ deinterlacer_v1_01_a /src
  This directory contains the source code of the pCore's software driver.

Name	Description
xdeint.c	Provides the Application Program Interface (API) access to all features of the Video Deinterlacer device driver.
xdeint.h	Provides the API access to all features of the Video Deinterlacer device driver.
xdeint_g.c	Contains a template for a configuration table of Video Deinterlacer core.
xdeint_hw.h	Contains identifiers and register-level driver functions (or macros) that can be used to access the Video Deinterlacer core.
xdeint_intr.c	Contains interrupt-related functions of the Video Deinterlacer device driver.
xdeint_sinit.c	Contains static initialization methods for the Video Deinterlacer device driver.

General Purpose Processor Files

When the interface selection is set to General Purpose Processor, CORE Generator then outputs the core as a netlist that can be inserted into a processor interface wrapper or instantiated directly in an HDL design. The output is placed in the <project directory>.

File Details

The CORE Generator software output consists of some or all the following files.

Name	Description
<component_name>_readme.txt	Readme file for the core.
<component_name>.ngc	The netlist for the core.
<component_name>.veo <component_name>.vho	The HDL template for instantiating the core.
<component_name>.v <component_name>.vhd	The structural simulation model for the core. It is used for functionally simulating the core.
<component_name>_synth.v <component_name>_synth.vhd	Synthesis instantiation wrapper file.
<component_name>.xco	Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_flist.txt	A text file listing all of the output files produced when the customized core was generated in the CORE Generator software.
<component_name>.asy	IP symbol file

Name	Description
<component_name>.gise	
<component_name>.xise	ISE® software subproject files for use when including the core in ISE software designs.

Chapter 4

Designing with the Core

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

Deinterlacing

The deinterlacer contains two processing kernels: the motion adaptive and the diagonal detection and adaptation processing kernels. These kernels work together to form each deinterlaced pixel.

The motion adaptive kernel has two threshold parameters that can be adjusted by the user if required. These two parameters are T1 and T2. They are used as threshold points for measuring between no motion, average motion, and excessive motion. In each of these categories, the deinterlacer generates the output pixels using different techniques. Figure 4-1 shows the conceptual relationship of the T1 and T2 parameters to the deinterlacer pixel creator.

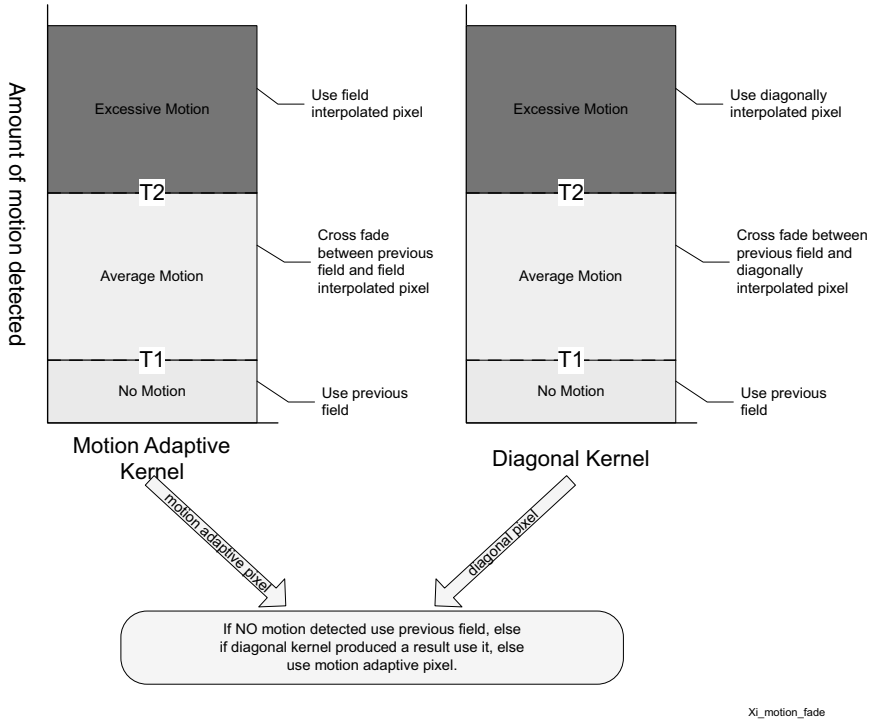

Figure 4-1: **Output Pixel Decision Criteria**

T1 and T2

T1 and T2 can be set to these default values:

Typical SDI YUV defaults: T1 = 10, T2 = 70

Typical SDI RGB defaults: T1 = 100, T2 = 200

Generally, they should not be altered, but users can alter them depending on the noise level of the input video signal. If the input video source is noisy, this may be detected as excessive motion and the output image may be of lower quality. In this case, the motion detection threshold can be increased by the application software. In this case, the motion detection threshold can be increased by application software.

Cross Fade Ratio

The cross fade scale register is derived directly from T1 and T2 according to this fixed equation:

xfade ratio = (4096*256)/(T2-T1)

This value is used internally to control cross fading between kernel pixels and the frame store pixels. This register must be changed whenever T1 or T2 are altered to ensure the correct operation of the cross fader.

Initial State

The deinterlacer kernel must have two fields of video history to produce its desired output. During a video input standard change, start-up condition, change of format or error state, there is no video history for the deinterlacer to use. For these frames (if enabled via software), the deinterlacer produces progressive video outputs without the aid of the motion adaptive kernel. As a result, these initial frames appear softer in format until the memory interface has obtained sufficient history for producing the required output quality.

Figure 4-2 illustrates the sequencing of the deinterlacer output with respect to input variance. The diagram shows the two initial frames (1 and 2) being created from raw passing video and then the remainder being produced with the aid of the historical data.

The second image shows a normally operating deinterlacer that is suddenly subjected to a change in input video. The deinterlacer then resets the memory interface and reverts to a lower quality, while it builds up new picture history over the first two frames. It then reverts to fully operational state.

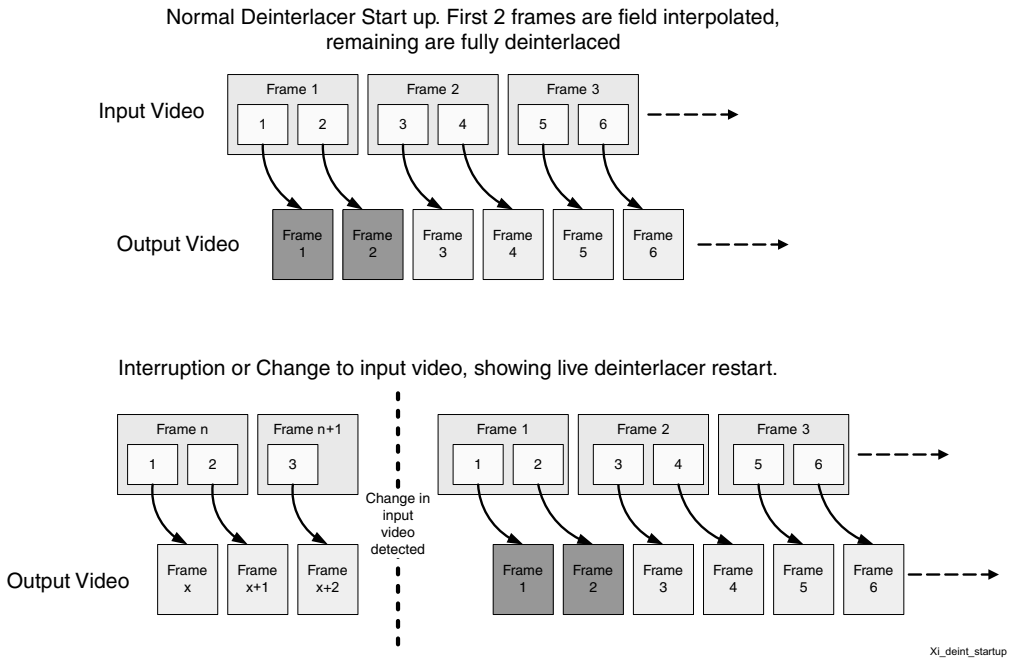

Figure 4-2: **Examples of Deinterlacer Start-up Conditions**

Architecture

The Xilinx Video Deinterlacer converts a live input video stream into a progressive video structure. Figure 4-3 illustrates a high-level view of the ports of the deinterlacer.

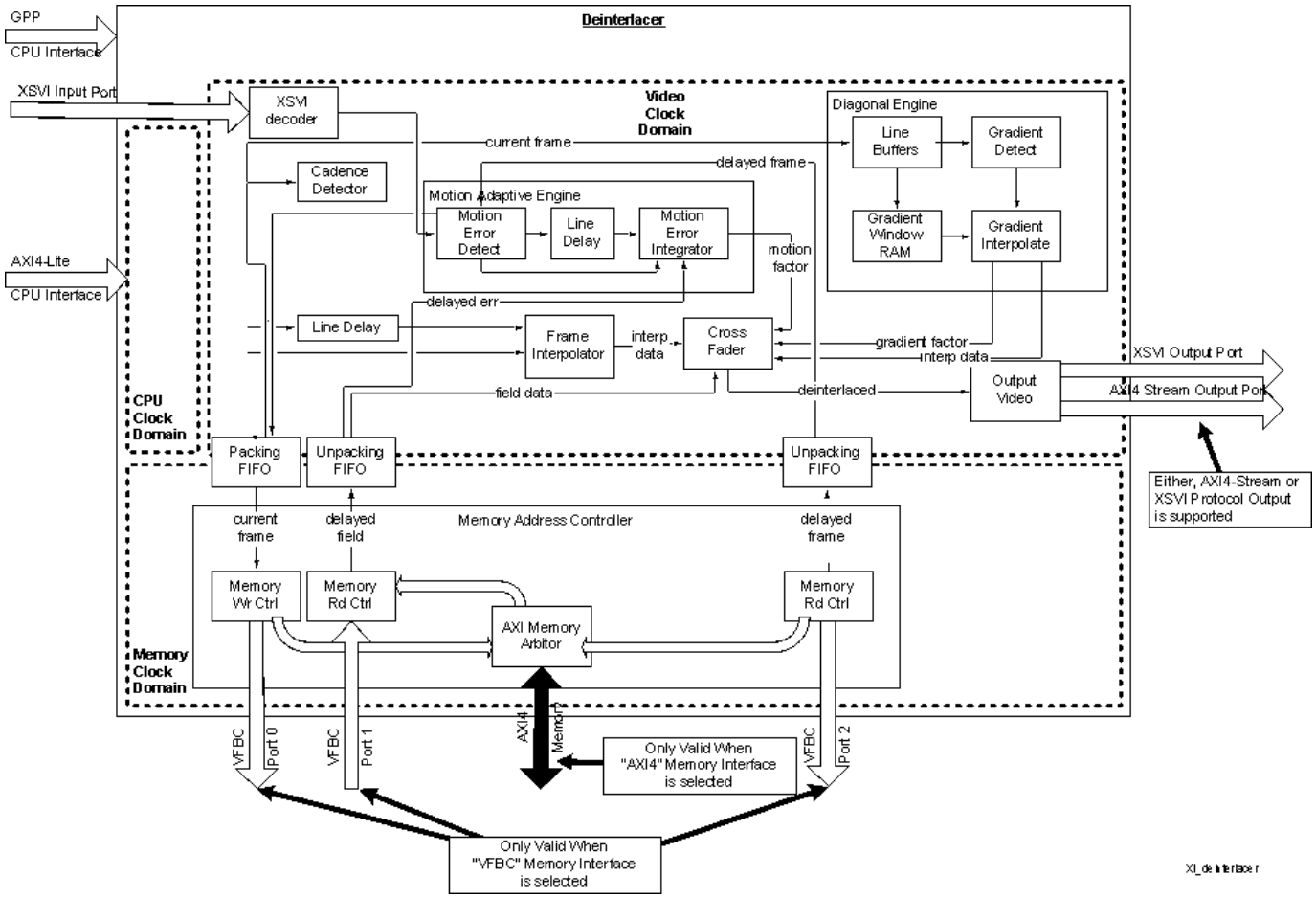

Figure 4-3: **Deinterlacer Data Path Overview**

In conjunction with the video path, the VFBC or AXI4-MM ports read and write passing video fields to and from a memory buffer. These fields of information are used by the deinterlacer internal processing blocks to produce the final progressive video output.

In creating progressive pictures, the output frame rate of the deinterlacer is always twice the input rate and produces one pixel per clock. The video clock used must meet this system requirement. The input pixel rate must be less than or equal to the video clock rate divided by two. The output pixel rate is always twice the input pixel rate. A single common video clock is used for the entire video path.

The video deinterlacer input can be either from live video or a stored video feed. The Xilinx Streaming Video Interface (XSVI) input bus is clock enabled to allow for continuous or burst input rates. An optional full flag allows for push back of input data when the deinterlacer is receiving input from a non-live video feed. The XSVI output bus is also clock enabled and produces output pixels whenever there is a pixel inside the deinterlacer to be generated. The video deinterlacer has only minimal buffering inside. It is important to not overflow the input FIFO.

Memory Controller

The deinterlacing process requires two previous fields of video information to determine the amount of per-pixel motion present in the passing video. It then selects the most appropriate method of deinterlacing each pixel using these streams.

An external memory store is used in a triple buffer concept to store and extract passing video fields and associating sideband data. At the end of each output frame, the memory controller moves its base pointers to the next buffer and starts again. Figure 4-4 illustrates the triple buffer movement:

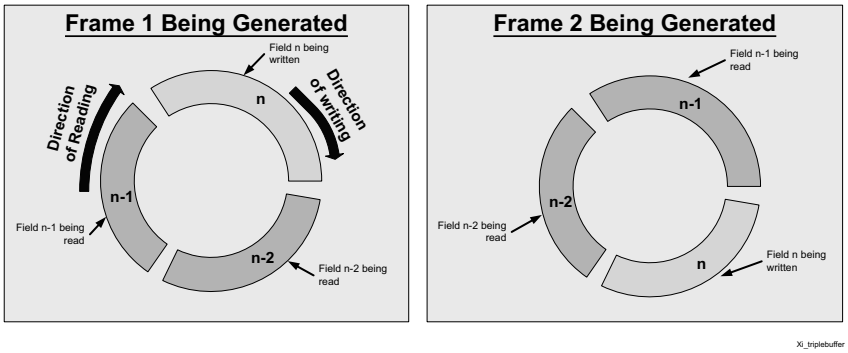

Figure 4-4: **Triple Buffer Usage**

The memory ports operate in a unidirectional manner, 1 write and 2 read. It continuously stores the incoming field with its motion vector and extracts fields n-1 and field n-2 from the other two buffers.

To provide efficient memory utilization, the pixel stream and error stream are tightly packed into the VFBC or AXI4-MM data streams. Depending on the configured bit depth, there are three different packing formats. The stored video image should not be used by other modules. This information is an internal memory pool, although it can be monitored if needed.

Figure 4-5 illustrates the memory packing algorithm. Fields marked "pix" indicate 444 pixels and fields marked "err" are the associated motion error vector.

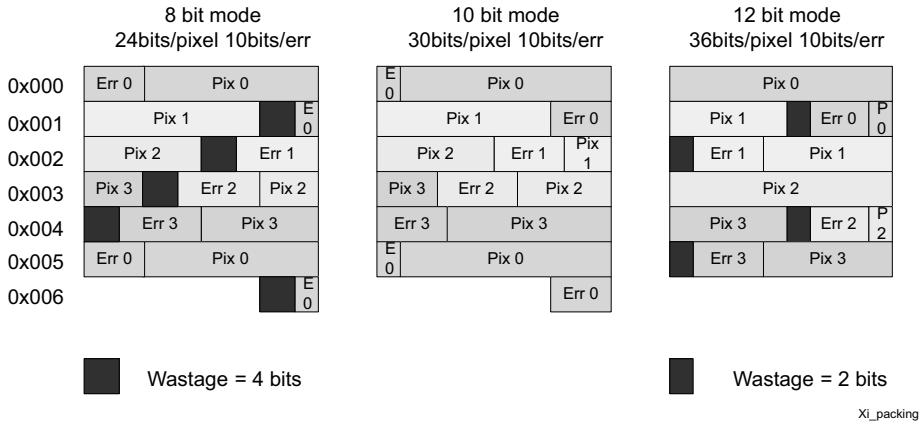

Figure 4-5: **VFBC Data Packing Format**

Memory Size

When calculating memory requirements for the deinterlacer, the packing method and input video field size must be considered. For 8 and 10-bit color depth, the ratio is (5/4) because five words are required to store four pixel/error pairs. For 12-bit color depth, the ratio is (3/2) because three words are required to store two pixel/error pairs. For example:

8-bit image with 720 wide requires : 720 * (5/4) dwords = "900" per line

12-bit image with 1920 wide requires: 1920 * (3/2) dwords = "2880" per line

Consequently, for a full 12-bit image that is 1920 wide and 540 lines per field, a buffer of 2880 * 540 = 1.55 Mwords = 6.22 Mbytes is required. The total for the triple store is 18.7 Mbytes of storage.

Note: The ratios of 5/4 and 3/2 impose a line width limitation on the deinterlacer. The number of dwords per line must result in an integer value. For example, this would not be allowed: 719 8-bit pixels = 719* (5/4) = 898.75.

Consequently, for 8 or 10-bit images, the xsize parameter must be divisible by 4, and for 12-bit images, the xsize parameter must be divisible by 2.

I/O Interface and Timing

AXI4-Lite and GPP Interface

When selected via CORE Generator, an AXI4-Lite interface is included in the IP module. This interface is used to configure the deinterlacer dynamically during run time. While the interface operates in its own clock domain, the transfer of register information into the deinterlacer and memory controller is done synchronously. All registers are shadowed in their respective domains.

There are three categories of registers inside the core:

- Global Registers

  Located in the AXI4-Lite clock domain and used internally by the deinterlacer for core wide operations, including forcing modes and completely disabling the deinterlacer.

- Deinterlacer Configuration Registers

  Used to specify most of the aspects in deinterlacing, including algorithm selection, threshold control, raster size, color space and so on.

- Memory Controller Configuration Registers

  Used to set up the triple field buffer memory regions that are required by the deinterlacer core.

Dynamic Reconfiguration

When working with multiple input standard streams that can change from frame to frame, the deinterlacer can transition smoothly from one format to the next without producing any unnecessary data at its output. This is achieved through the AXI4-Lite interface scheduler.

When system software programs the AXI4-Lite registers, only registers within the AXI4-Lite domain are affected. These registers can be freely written to or read from. After the software has committed to a new configuration, it writes to the global register and asserts an update request.

After this request is queued, all of the deinterlacer registers become read-only (apart from the global register). Upon the next frame boundary, the deinterlacer shadows all registers and begins processing using the new settings. This synchronous transfer ensures a clean transition from one format to the next.

If the software decides to stop the update request, it can cancel it using the global register. This operation occurs immediately as a force operation and should generally not be used under normal operating conditions. The disabling can occur coincident with the actual internal update and can cause the deinterlacer to generate unnecessary output.

Interrupts

The deinterlacer core provides eleven interrupt events to ensure efficient use of the system AXI4-Lite when using a deinterlacer. All interrupts have their own status register and can be independently enabled, disabled, and cleared. Under normal operating conditions, the deinterlacer does not require AXI4-Lite interaction. However, interrupts can be used to aid in monitoring the system state.

These interrupts are:

- Internal register update has occurred

  Used to acknowledge the register update request event.
- Deinterlacer synchronized

  Indicates input video raster is stable and matches programmed x/y sizes known to deinterlacer.
- Deinterlacer has lost synchronization

  Indicates different input video raster to programmed x/y/ sizes, or input is not stable.
- Deinterlacer internal FIFO over run error

  Occurs if video clock is not fast enough to process input video.
- Pull down controller is activating

  Indicates that a pull-down cadence is detected and output video is now derived by the cadence.
- Pull down controller is deactivating

  Indicates that the pull-down controller has detected the disappearance of the cadence, and the deinterlacer is reverting to normal mode.

AXI4-Lite Timing

The AXI4-Lite interface is used for programming the video deinterlacer operational modes and interrupt system. Read or write accesses to the AXI4-Lite port are considered low bandwidth and as such the slave port only processes one AXI4-Lite access at a time. If the deinterlacer is presented with a simultaneous read and write operation, the write operation takes precedence and the read operation stalls. Once the write operation is complete, the read operation completes.

Figure 4-6 shows several write operations followed by several read operations and illustrates the read and write timing of the AXI4-Lite interface.

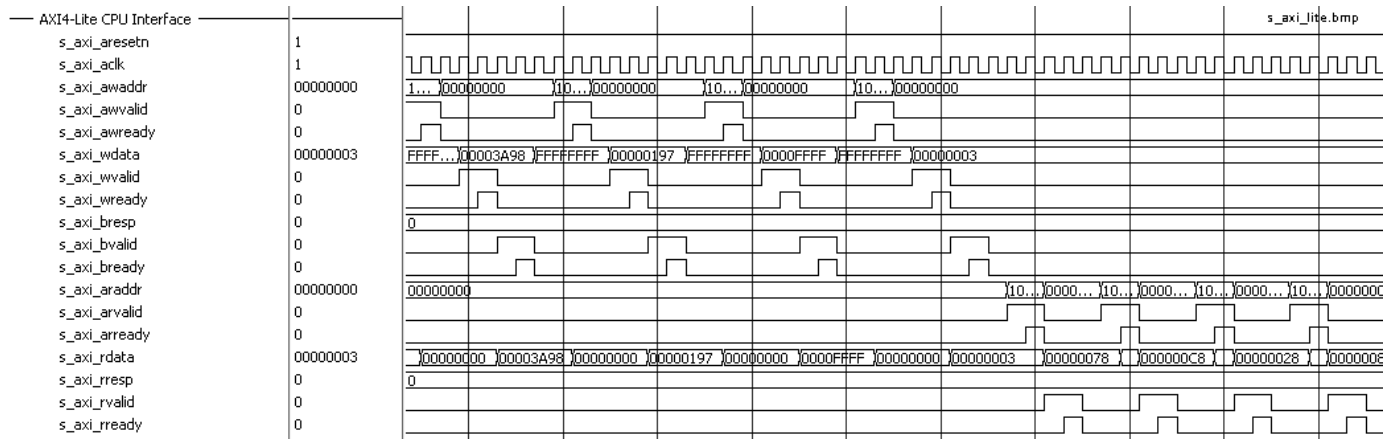

Figure 4-6: **AXI Slave Write and Read Operations**

All AXI4-Lite signals not required by the AXI4-Lite specification have no connection to the deinterlacer.

General Purpose Processor (GPP) Interface

The video deinterlacer can be configured to have a simple GPP interface instead of an AXI4-Lite port. This interface exposes all the GPP accessible registers for direct control by the system designer. The exception is that an interrupt mechanism is not available. The target system must monitor the status lines on the GPP to determine if any failures or faults have occurred.

The GPP signals are synchronous to the GPP clock domain. The internals of the deinterlacer operate using several clock domains. The same update mechanism for internal registers is used by the GPP port. The controlling system must prime the GPP signals on the GPP clock domain and then raise the "update request" input line. The deinterlacer then updates its internal registers on the next available frame interrupt and responds by raising the "update done" GPP output signal. The user should then lower the "update request" signal. This technique allows for the synchronous changing of deinterlacer operations resulting in an output that is always glitch free.

Control Interface

There are two control interface options available in CORE Generator software; EDK pCore or GPP. The interface types differ primarily in the method of delivery of the user-defined control values.

Control Values

In both GPP and pcore modes, the control values are provided dynamically at the input of the deinterlacer and can be changed during run time.

For the pcore version of the core, CORE Generator software provides the GPP core placed in a wrapper, which allows you to parameterize the deinterlacer core in EDK. The ports are driven by registers on a AXI4-Lite bus. The address is decoded in the wrapper. A MicroBlaze™ processor software driver is provided in source code form to drive these ports.

The parameters that can be set dynamically via AXI4-Lite registers are:

- packing: controls the YUV packing mode used; 4:2:2, 4:2:0 or 4:4:4
- kernel mode: controls what deinterlacer algorithms are used
- threshold T1: controls the low motion threshold
- threshold T2: controls the high motion threshold
- cross fade ratio: controls the scaling factor used by the cross fader
- xsize, ysize: controls the active window size of the output video frame
- field order: sets the field order as: HD,PAL or NTSC
- color: selects which color space is processed, YUV or RGB
- black: sets the pixel value for black inside the core, dependent on color space setting
- fswords: set the amount of 32-bit words that are required to store one field of video in the external memory buffer
- fsbase0,1,2: sets the 32-bit base addresses of the three external field buffers
- PsF mode: controls if the deinterlacer is processing interlaced, PsF or progressive image structures
- pull-down mode: controls if the pull-down controller is activated

Memory Interface

The video deinterlacer motion kernel requires video frame history to deinterlace the input video stream. The input video stream is processed and stored into an external memory store along with specific associating sideband information. The external memory store is then used in the reconstruction of the output video stream.

The memory controller splits up external memory into a rolling three video-field store, where one field is written to while two fields are read from. This triple field buffer is controlled autonomously by the deinterlacer and driven through the VFBC streams.

The AXI4-Lite interface allocates the base addresses of the three field buffers and the physical size of a buffer. System software can dynamically alter this on the fly if required to adapt to changing video formats.

The memory interface runs in its own clock domain. The clock rate of this interface must run at a slightly higher rate than the video interface clock. The bandwidth requirements of the memory interface are discussed in VFBC/MPMC Memory and Interface Option.

AXI4 Memory and Interface

The key features of the AXI-MM port are:

- Single port to move all 3 deinterlacer streams, reducing AXI-interoconnect overhead
- Asynchronous clock to Deinterlacer video path, allowing AXI clock to match interconnect to ensure highest efficiency bursting.
- Mutli thread support. To allow multiple data streams to move across a common bus
- Multiple outstanding requests. To reduce system latency impacts
- Scalable from 32 to 256 bits wide.

The AXI4-MM port stores and extracts video fields and error information used by the deinterlacer core. The AXI4-MM port operates in a multi-threaded bi-directional manner.

The internal Deinterlacer has 3 independent data streams all moving the internal packed data format. These streams comprise of one write stream and two read streams.

To further provide efficient memory utilization, the pixel stream and error stream are packed into the AXI data streams. Depending on the configured bit depth, there are three different packing formats.

Figure 4-7 illustrates the memory packing algorithm. Fields marked "pix" indicate 444 pixels and fields marked "err" are the associated motion error vector.

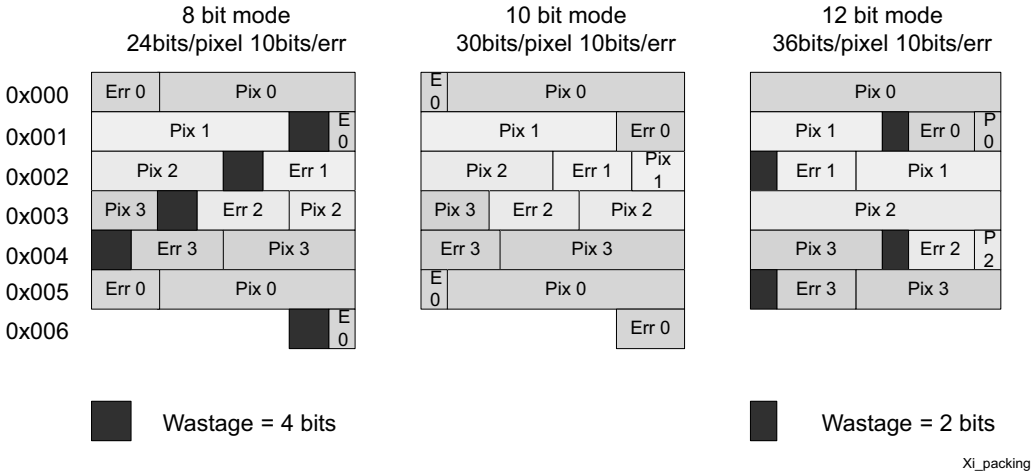

Figure 4-7: **VFBC Data Packing Format**

When calculating external memory requirements for the deinterlacer, the packing method and input video field size must be considered. For 8 and 10-bit color depth, the ratio is 5/4 as 5 words are required to store 4 pixel/error pairs. For 12-bit color depth, the ratio is 3/2 as 3 words are required to store 2 pixel/error pairs.

For example:

An 8-bit image with 720 wide requires : 720 * (5/4) dwords = "900" per line

A 12-bit image with 1920 wide requires: 1920 * (3/2) dwords = "2880" per line

Write Stream

The AXI memory controller uses the AXI-Write channel to push all write data onto the AXI-interconnect at the configured data-width given. All bursts are a fixed length of 32 beats in length (m_axi_awlen). Thus for wider data bus widths more data is conveyed per burst.

All write operations ensure highest bus efficiency with back-to-back data packing and no narrow transactions. The Deinterlacer will only request a AXI transaction if it has data to immediately move.

The write stream will only generate 1 outstanding transaction at a time. A typical burst is shown below of beat length 0x1F, to address 0x41700E00. The initial queing of the burst can be seen, followed by a continuos of 32 beat burst of data. Whilst "m_axi_wvalid" is constantly high, the "m_axi_wready" pushback from the AXI-interconnect is demonstrating possible throttling by a downstream memory controller.

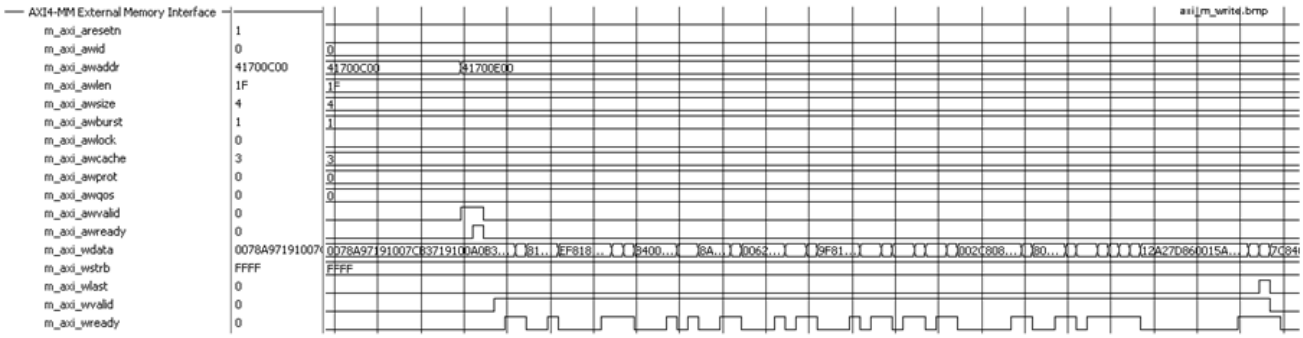

Figure 4-8: **Write Stream Burst**

Read Stream

The AXI memory controller uses the RD channels of AXI to extract 2 streams of video information from the external memory interface. To ensure efficient use of the AXI bus and external memory controller, the deinterlacer's memory controller uses :

- Multiple outstanding reads to ensure system latency's have no impact on the deinterlacer processing.
- Multiple thread-id's to all for 2 read-streams to share a single common AXI port.

Any downstream memory controller must be configured to support the above features. The Xilinx AXI-Memory controller can easily be configured for such a usage model.

To ensure no wasted AXI bandwidth or interconnect throttling occurs, the Deinterlacer will only issue read requests if it can fully accept the read data. The read-ready strobe is permanently tied high (m_axi_rready).

All bursts are a fixed length of 32 beats in length (m_axi_arlen). Thus for wider data bus widths more data is conveyed per burst. No narrow bursting is done

Each of the 2 streams are given a static unique AXI "thread-id", these being 0 & 1.When transactions are posted onto the AXI-interconnect, the downstream module will maintain a list of the id's of each request and return the id alongside the returning data burst. The Deinterlacer then routes the inbound data to the correct internal read stream.

In order to cater for unpredictable system latencies the Deinterlacer per thread-id issues up to 2 outstanding read request. A maximum of 4 outstanding requests can be seen in systems with high read latency, and the target memory control should be configured to support this mode of operation.

Shown below is a multi-threaded read operation, the diagram is highlighted to indicate thread 0 and 1's independent read requests, followed by the returning data (tagged with the correct id) The diagram also illustrates an external memory controller that is unable to fully supply data to the axi-interconnect at its line rate, and thus m_axi_rvalid is toggling throughout the read data bursts.

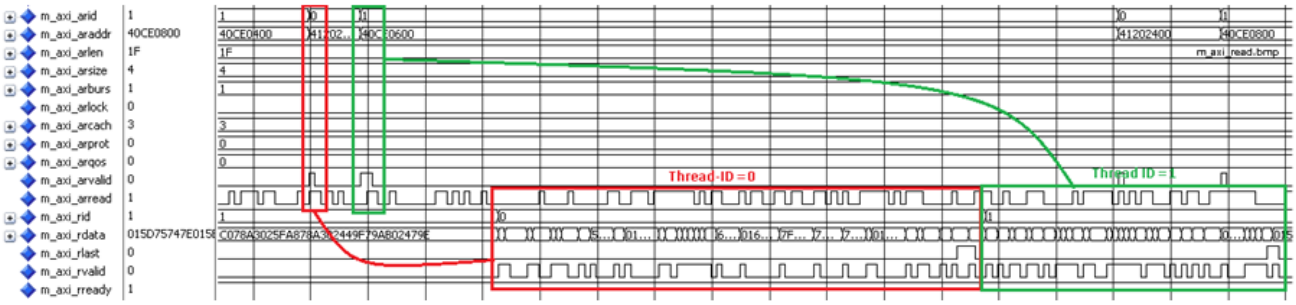

Figure 4-9: **Read Stream Burst**

Clocking

There is a minimum clock requirement on the AXI clock (vfbc_clk). The AXI-MM domain must provide the deinterlacer with its data in a timely manner. This requirement combined with the packing formats inside the AXI controller and the data width of the AXI-MM bus yield a minimum clock rate.

The formulas below are theoretical minimums that assume the read and write streams can process data with 100% efficiency. If the system cannot achieve this, the AXI clock rate should be scaled accordingly to cater for the correct system efficiency.

The base formula is:

write_32bit_words_second = packing ratio * pixel rate

read_32bit_words_second = 2 * packing ratio * pixel rate

axi_clk= read_32bit_words_second*(axi_data_width/32)

Shown below is a selection of examples of the above equations.

AXI Clock Rate	Pixel Rate	Packing Ratio	Reads/Sec	Writes/Sec	AXI Data Width
33.75MHz	(SD) 13.5MHz	8bit = (5/4)	33.75MHz	16.875MHz	32bits
185.6MHz	(HD) 74.25MHz	8bit = (5/4)	185.6MHz	92.8MHz	32bits
46.4MHz	(HD) 74.25MHz	8bit = (5/4)	185.6MHz	92.8MHz	128bits
111.3MHz	(HD) 74.25MHz	12bit = (3/4)	222.75MHz	111.3MHz	64bits

1. The signal is named vfbc_clk instead of the typical m_axi_aclk. This is due to the clock pin being shared with either the VFBC or AXI implementations.

VFBC/MPMC Memory and Interface Option

The VFBC ports store and extract video fields and error information used by the deinterlacer core. Each VFBC port operates in a unidirectional manner, with port 0 as a write port and ports 1, 2 as read ports.

To provide efficient memory utilization, the pixel stream and error stream are packed into the VFBC data streams. Depending on the configured bit depth, there are three different packing formats.

Figure 4-7 illustrates the memory packing algorithm. Fields marked "pix" indicate 444 pixels and fields marked "err" are the associated motion error vector.

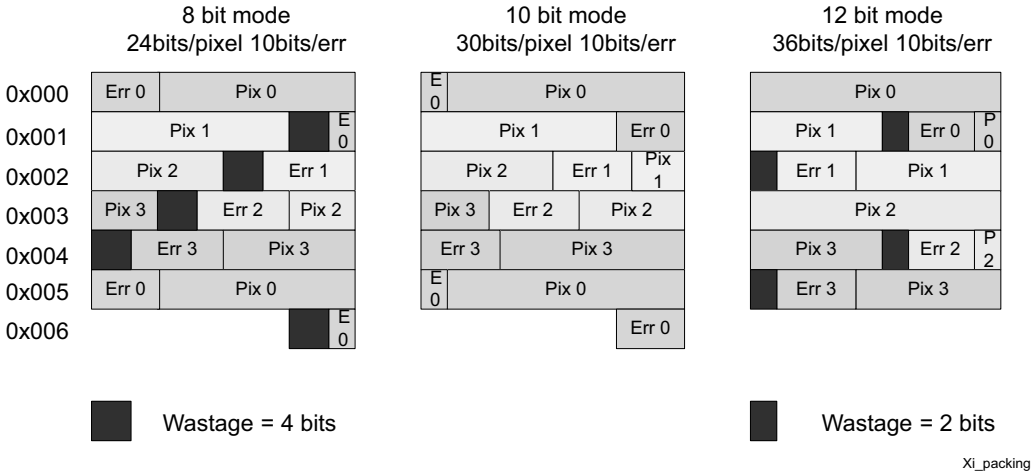

Figure 4-10: **VFBC Data Packing Format**

When calculating memory requirements for the deinterlacer, the packing method and input video field size must be considered. For 8 and 10-bit color depth, the ratio is 5/4 as 5 words are required to store 4 pixel/error pairs. For 12-bit color depth, the ratio is 3/2 as 3 words are required to store 2 pixel/error pairs.

For example:

An 8-bit image with 720 wide requires : 720 * (5/4) dwords = "900" per line

A 12-bit image with 1920 wide requires: 1920 * (3/2) dwords = "2880" per line

VFBC Clocking

There is a minimum clock requirement on the VFBC clock. The VFBC domain must provide the deinterlacer with its data in a timely manner. This requirement combined with the packing formats inside the VFBC controllers give a minimum clock rate. The formula is:

vfbc clock = packing ratio * pixel rate

For example, using a triple port, given an input pixel rate of 13.5 MHz (SD Video) and 8-bit color depth, the VFBC Clock must run at least (5/4) * 13.5 MHz = 16.875 MHz so the VFBC can sink and source data fast enough. Due to memory latencies, a sensible safety margin of 50% above the calculated minimum is recommended. This 50% overhead leaves a minium memory clock rate of 34 MHz.

VFBC Write Stream

Each field of data is pushed into the VFBC interface as data passes into the XSVI input bus. The VFBC write controller breaks up the passing video frame into bursts of 128*32bit words. Each burst is then transmitted across the VFBC port. For video frame sizes that do

not end at a 128 words boundary, the VFBC write controller still generates the 128 word burst and pads the remaining data.

Figure 4-11 illustrates the VFBC Write Stream processing SD-Video. A short burst of write data and overlapping command access occur. The VFBC Write stream maintains at least one command queued in advance, so that the data stream can be kept continuous.

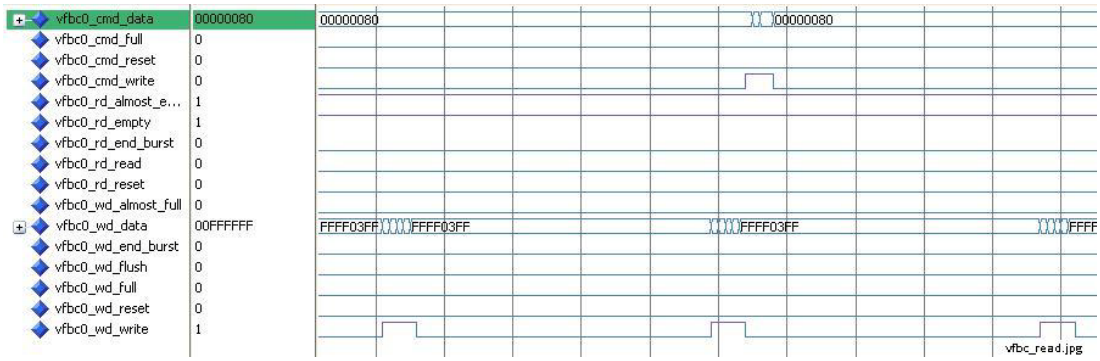

Figure 4-11: **VFBC Write Stream**

VFBC Read Stream

Each field that passes through the deinterlacer requires two fields worth of history to be extracted from the memory interface. Two VFBC read streams extract this information in bursts of 128*32bit words. Each burst is then stored in a deinterlacer FIFO ready for processing by the deinterlacer kernels. For video frame sizes that do not end at a 128 word boundary, the VFBC read controller generates a 128 word burst and then abandons the extra data.

To provide the data in a timely manner, both read queues are always primed with an outstanding command. This enables the memory controller to immediately move onto the next read burst after it has completed the previous burst.

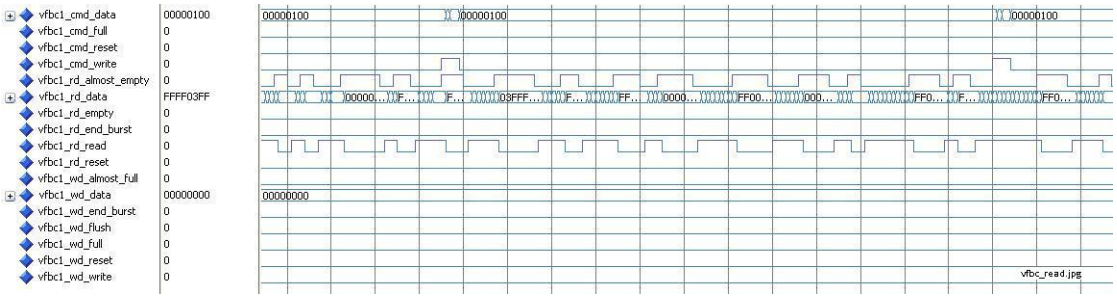

Figure 4-12: **VFBC Read Stream**

Video Interface

The video deinterlacer has one input and output video port. The input video timing (hblank/vblank) is used solely to identify the first pixel of each input frame. The specific width of the horizontal and height of vertical blanking intervals are not significant but must have a minimum width of one video clock pulse.

The deinterlacer only processes the active video portion of the input video, all other blanking data is discarded. Critically, the core generates pixels at twice the input rate of

input video data. To ensure the system can process the input data without loosing pixels, a core wide video clock-enable strobe is provided.

This must be used on the input side to throttle the input video to half the video clock rate. The waveform of the clock enable must only maintain an average of 50% active, the period of this signal can be random. Figure 4-13 illustrates an example video clocking of the deinterlacer.

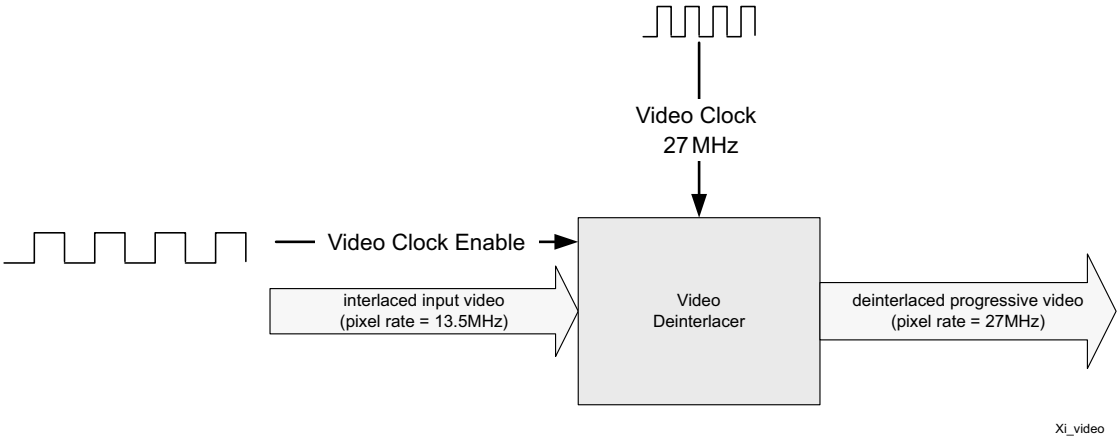

Figure 4-13: **Input to Output Video Clock Ratio for SD**

The core output is always progressive in format when the deinterlacer is enabled and a synthetic video timing frame is constructed around the output stream to provide vertical and horizontal blanking strobes for downstream cores.

Figure 4-14 illustrates typical input and output frame structures.

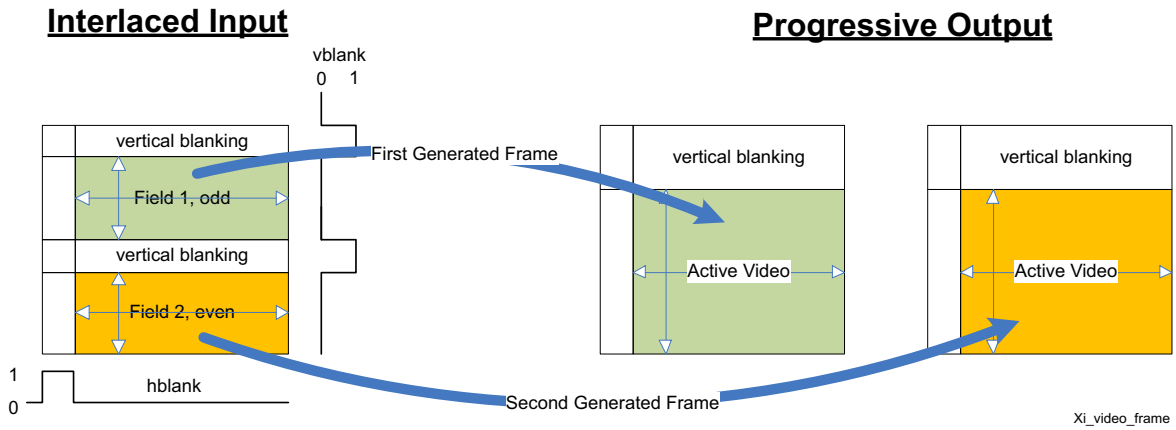

Figure 4-14: **Input and Output Video Timing Formats**

The video deinterlacer can process either 4:2:0, 4:2:2 or 4:4:4 video formats. These can either be statically set at core configuration time or can be configured to be dynamically controllable by system software.

Figure 4-15 illustrates the video timing of the various supported packing formats.

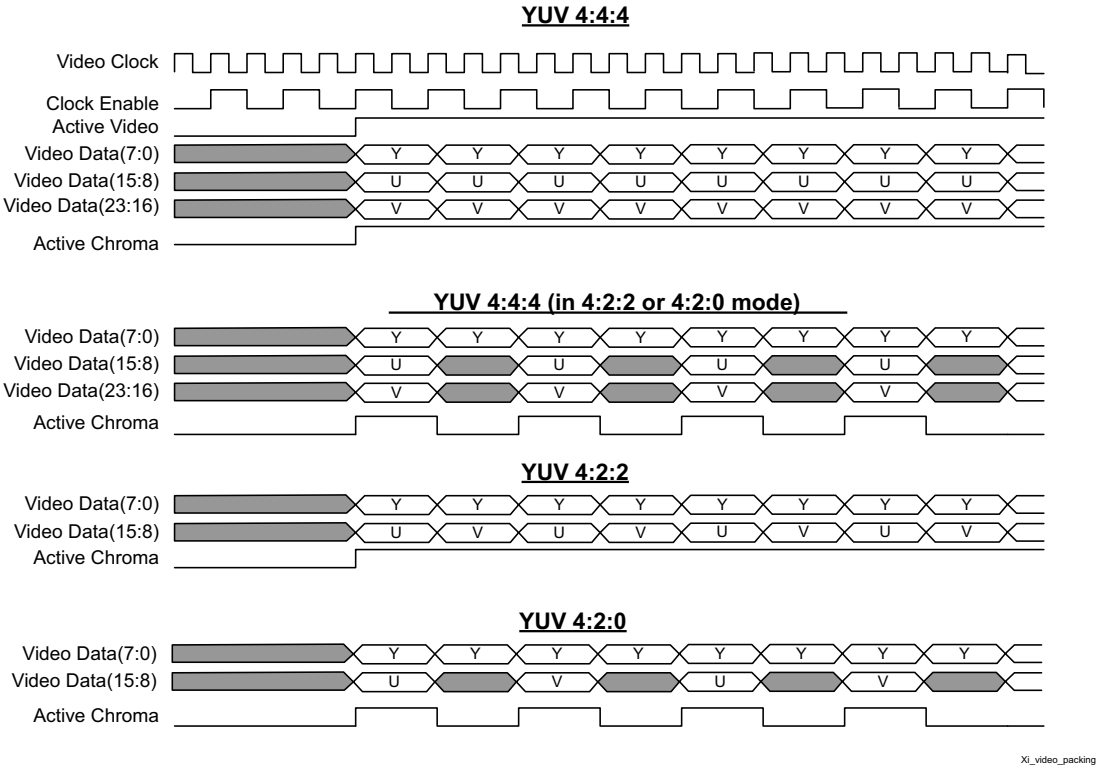

Figure 4-15: **XSVI Input and Output Packing Formats**

When using the motion kernel, the deinterlacer must have two fields of video history to create the desired output. During a video input standard change, start-up condition or error state, there is no video history for the deinterlacer to use. For these frames, the deinterlacer produces progressive video outputs without the aid of the motion adaptive kernel. Consequently, these initial frames appear softer in format until the memory interface has obtained sufficient history so that it can produce the required output quality.

Figure 4-16 illustrates the sequencing of the deinterlacer output with respect to input variance. The diagram shows the two initial frames (1 and 2) being created from raw passing video and then the remainder (3-7) being produced with the aid of the historical data.

The second image shows a normally operating deinterlacer that is suddenly subjected to a change in input video. The deinterlacer then resets the memory interface and reverts to a

lower quality, while it builds up new picture history over the first two frames. It then reverts to a fully operational state from then on.

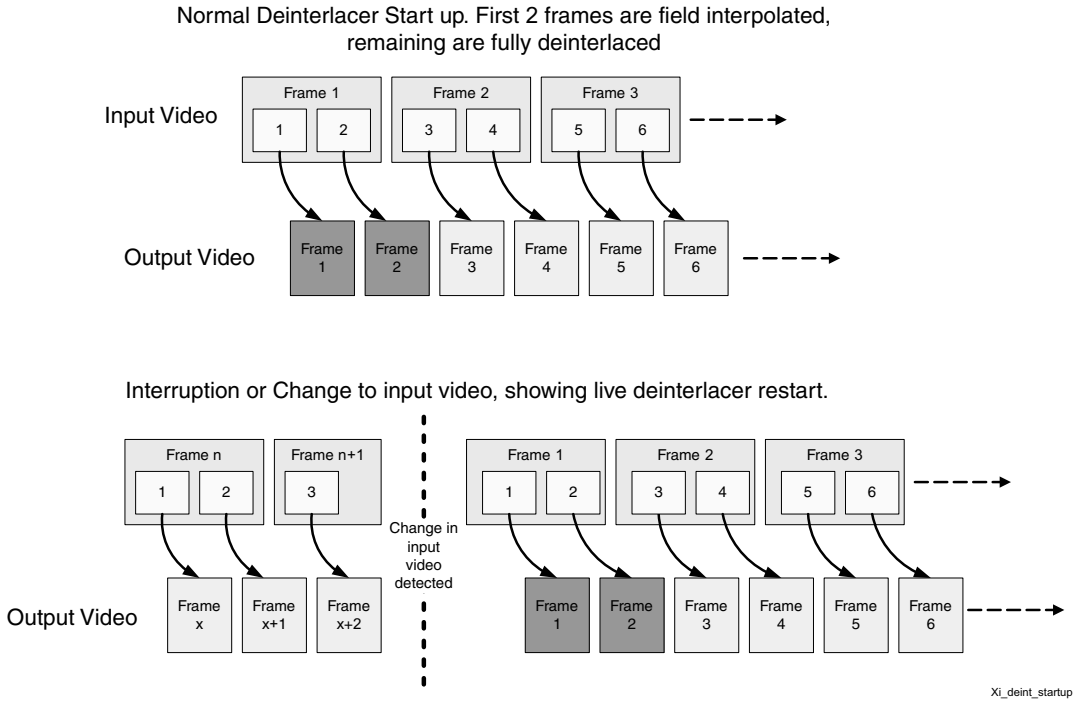

Figure 4-16: **Examples of Deinterlacer Start-up Conditions**

Clocking

To provide a compact design, the deinterlacer provides only minimal buffering required in performing the deinterlacing operation. Extra buffering required by the use of the full/pause-flags as system push back are outside the scope of this module.

The Video Deinterlacer comprises these clock domains:

- Video Clock Domain: All video passes through this common clock domain and the deinterlacer core resides here.
- AXI4-Lite Clock Domain: The AXI4-Lite interface and interrupt signalling operates on its own exclusive domain.
- Memory Clock Domain: All memory ports use a common clock that is exclusive to the memory interface(s).
- The user can combine or keep these clock domains separate as per their architecture requirements.

Resets

The Video Deinterlacer core has multiple reset inputs, one for each clock domain. The Video Deinterlacer core comprises these reset inputs.

- Video Clock Domain: sclr (active high)
- AXI4-LiteClock Domain: s_axi_aresetn (active low)
- Memory Clock Domain: m_axi_aresetn (active low)

Protocol Description

The Video Deinterlacer core register interface is compliant with the AXI4-Lite interface. The memory interface is compliant with the AXI4 Memory Mapped interface. The Video Deinterlacer output interface can be configured to be compliant with the AXI4-Stream interface.

XILINX®

Chapter 5

Constraining the Core

Required Constraints

There are no required constraints for the Video Deinterlacer core.

Device, Package, and Speed Grade Selections

There are no Device, Package or Speed Grade requirements for the Video Deinterlacer core.

Clock Frequencies

There are no specific clock frequency requirements for this core.

This core has not been characterized for use in low power devices.

Clock Management

The Video Deinterlacer core has 3 clock inputs: `vid_clk`, `vfbc_clk`, and `s_axi_aclk`. The `vid_clk` is used for core processing. The `vfbc_clk` is used for the input/output VFBC or AXI Memory Mapped interfaces. The `s_axi_aclk` is used for the AXI4-Stream output interface. All 3 clock domains can be considered asynchronous to each other. No relationship is required.

Clock Placement

There are no specific Clock placement requirements for this core.

Banking

There are no specific Banking rules for this core.

Transceiver Placement

There are no Transceiver Placement requirements for this core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for this core.

Chapter 6

Detailed Example Design

The deinterlacer is typically used in the broadcast video conversions of SD and HD material to progressive formats for subsequent display on a display monitor.

Case 1: SD480i to SD480p

One typical use of the deinterlacer is the fixed conversion of NTSC to 480p video. In this application, the deinterlacer uses the GPP interface and the configuration values are statically wired at the top layer of the final design. A CPU is not required for this implementation.

The core is configured to process an 8-Bit 4:2:2 YUV stream coming from a XSVI SDI input stream. The T1, T2 and cross fade ratio settings are wired to their default values. Full deinterlacing is enabled.

Given a pixel rate of 13.5 MHz for SD video, the video clock required is at least 27 MHz as shown in Figure 6-1. This can be derived from the incoming video and passed through a DCM to double the clock rate.

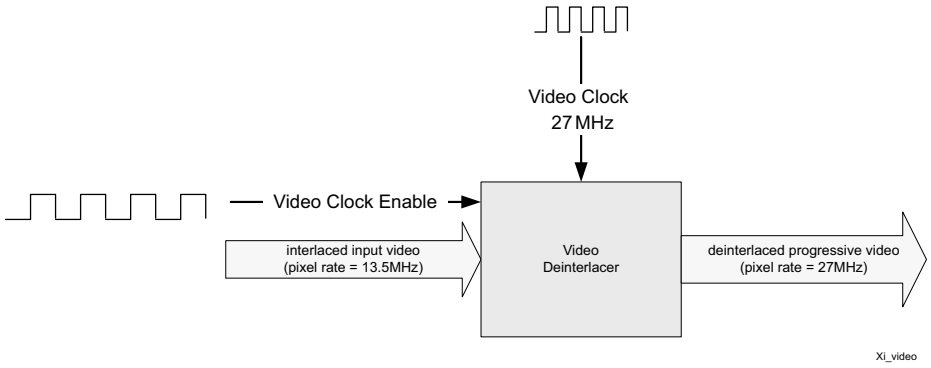

Figure 6-1: **Example SD Data Path**

The memory clock is set by considering the bit depth and pixel rate. Since 8-bit video is used, the packing ratio is 5/4. A safety margin of 70% VFBC utilization is used. Taking these factors into account, the minimum memory clock rate is:

Memory clock = [13.5 MHz * (5/4) *] / 0.70 = 24.1 MHz

The SDI system clock minimum is 13.5 MHz. Using a minimal clock approach, the video clock and memory clock can be connected and run at a common multiple of 13.5 MHz. 27 MHz is the first DCM multiple to satisfy the requirements of both the memory clock and video clock.

The memory bandwidth can now be determined. The deinterlacer has three memory streams, so the effective memory bandwidth of SD is:

24.1 MWords/Second * 3 streams = 72.3 MW/s or 289 Mbytes/s

Case 2: HD1080i to HD1080p

Another typical use of the deinterlacer is for the conversion of 1080iHD to 1080pHD video. In this application, the deinterlacer uses the AXI4-Lite interface and the configuration values are dynamically set by the system software.

The core is configured to process a 10-bit 4:4:4 YUV stream from a XSVI SDI input stream. The T1, T2 and cross fade ratio settings are set to their default values; full deinterlacing is enabled.

Given a pixel rate of 74.25 MHz for HD video, the video clock required is at least 148.5 MHz as shown in Figure 6-2.

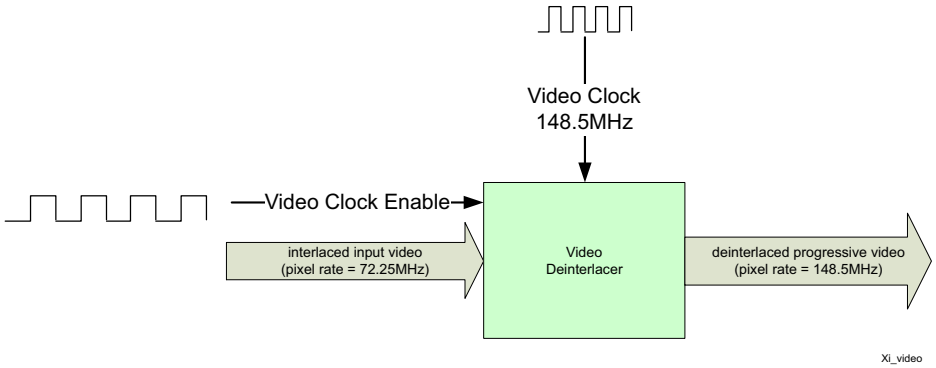

Figure 6-2: **Example HD Data Path**

The memory clock is set by considering the bit depth and pixel rate. Using 12-bit video, the packing ratio is 3/4, and with a safety margin of 60% VFBC utilization, the minimum memory clock rate is:

Memory clock = [74.25 MHz * (3/2)] / 0.60 = 185 MHz

The memory bandwidth can now be determined. The deinterlacer has three memory streams, so the effective memory bandwidth of SD is:

185 MWords/Second * 3 streams = 556 MW/s or 2.2 GBytes/s

For example, selecting a 32-bit DDR interface with a fabric clock rate of 200 MHz, physical clock rate of 400 MHz and DDR3-800 device, the theoretical bandwidth is 3.2 GBytes/s. This device configuration would sustain the deinterlacer, leaving 1 GB/s for other applications.

Directory and File Contents

- Expected
  - c_deinter0000.bmp
  - c_deinter0001.bmp
  - c_deinter0002.bmp
  - c_deinter0003.bmp

    - c_deinter0004.bmp
    - c_deinter0005.bmp
    - c_deinter0006.bmp
    - c_deinter0007.bmp
    - c_deinter0008.bmp
    - c_deinter0009.bmp
    - c_deinter0010.bmp
    - c_deinter0011.bmp
    - c_deinter0012.bmp
    - c_deinter0013.bmp
    - c_deinter0014.bmp
    - c_deinter0015.bmp
    - c_deinter0016.bmp
    - c_deinter0017.bmp
    - c_deinter0018.bmp
    - c_deinter0019.bmp
    - c_deinter0020.bmp
- Stimuli
    - FormulaOne_035.yuv
    - FormulaOne_036.yuv
    - FormulaOne_037.yuv
    - FormulaOne_038.yuv
    - FormulaOne_039.yuv
    - FormulaOne_040.yuv
    - FormulaOne_041.yuv
    - FormulaOne_042.yuv
    - FormulaOne_043.yuv
    - FormulaOne_044.yuv
    - FormulaOne_045.yuv
    - FormulaOne_046.yuv
    - FormulaOne_047.yuv
    - FormulaOne_048.yuv
- Results
- src
    - v_deinterlacer_v1_0_u0.vhd
    - v_deinterlacer_v1_0_u0.xco
- tb_src
    - axi_model.vhd
    - bmp_reader.vhd
    - bmp_writer.vhd

  - include_deint_tb.vhd
  - mpmc_model.vhd
  - testbench.vhd
  - vid_gold.vhd
  - vid_reader.vhd
  - vid_writer.vhd
  - yuv_writer.vhd
- isim_wave.wcfg - Waveform configuration file for iSim
- mti_wave.do - Waveform configuration for ModelSim
- run_isim.bat - Runscript for iSim in Windows OS
- run_isim.sh - Runscript for iSim in Linux OS
- run_mti.bat - Runscript for ModelSim in Windows OS
- run_mti.sh - Runscript for ModelSim in Linux OS

Demonstration Test Bench

A demonstration test bench is provided as a simple introductory package that enables you to observe the core generated by the CORE Generator tool operating in a waveform simulator. You are encouraged to observe core-specific aspects in the waveform, make simple modifications to the test conditions, and observe the changes in the waveform.

Simulation

- Simulation using ModelSim for Linux:
  From the console, Type **`source run_mti.sh`**.
- Simulation using ModelSim for Windows:
  Double-click on `run_mti.bat` file.
- Simulation using iSim for Linux:
  From the console, Type **`source run_isim.sh`**.
- Simulation using iSim for Linux:
  Double-click on `run_isim.bat` file.

Messages and Warnings

"Memory Collision Errors" have been observed when running the demonstration test bench. The issue has been investigated and it has been determined that these errors can be safely ignored. This error message can be suppressed in ModelSim when the global `SIM_COLLISION_CHECK` option is set to `NONE`.

XILINX®

Appendix A

Verification, Compliance, and Interoperability

Simulation

A validation suite consisting of a precompiled Windows C model and RTL test bench framework is included with the video deinterlacer. Both environments allow users to stream their own 24-bit true color BMP or YUV8/YUV10 files into the simulator and produce real BMP output files. This advantage allows for real world examples to be tested with the deinterlacer in advance. Additionally, an AVI video sequence file is also generated by the C model, allowing users to view animated results of the simulation in their chosen video program.

Additional system simulation and FPGA colorization is available via the AXI4-Lite interface to illustrate the algorithms operation and decision matrix in live operation. This can be useful if dynamic control of the thresholds is done by the system software. Figure A-1 shows a normal fully deinterlaced output. Note the smoothed lines of the deinterlacer.

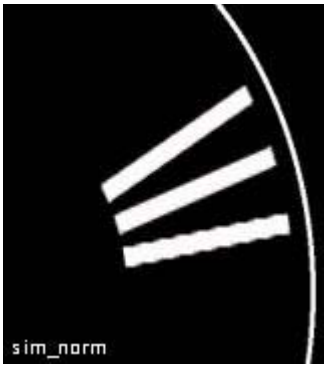

Figure A-1: **No Colorization**

Figure A-2 is the same image with full diagonal colorization enabled. The green highlight shows the diagonal edges that were detected and then enhanced.

Figure A-2: **Diagonal Colorization**

Figure A-3 is the same image with full motion colorization enabled. The three lines are moving upward. The three trailing motion vectors are in red around each white line. The red lines show the front and back edge motion of the line.

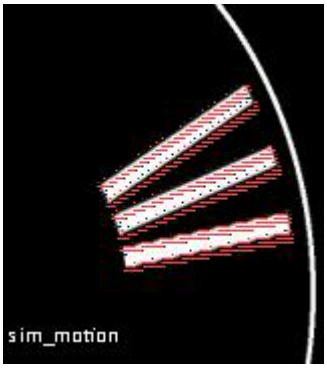

Figure A-3: **Motion Colorization**

Hardware Testing

Xilinx has a Real Time Video Engine reference design that has incorporated the Deinterlacer. The RTVE design targets both Spartan-6 and Virtex-6 FPGA platforms.

XILINX®

Appendix B

Debugging

Step 1: Video Pass Through Bring Up

When initially bringing up the Deinterlacer in a simulator or FPGA environment, the Deinterlacer can be configured to use minimal external interfaces. Use of the interrupt mechanism is strongly advised, as this gives a real time indication of possible system issues.

After system reset the Deinterlacer will start in bypass mode. This mode of operation requires no external memory interface for the Deinterlacer to move video through itself. It does require that the input and output video streams are operational.

By using the interrupt mechanism, the user can determine if the system is stable as no error interrupts will occur. At this point video should be passing through the Deinterlacer data path in its native format, (except all blanking will have been removed) System designers should observe the video output matches the video input.

If error interrupts occur, the likelihood is that either the input or output fifo's have over run.

- Input fifo overrun occurs if the output fifo is stalled for too long, or the `vid_clk` is not running fast enough.
- Output fifo overrun occurs if the output fifo is stalled for too long (>1000 clks)

Step 2: Basic Deinterlacing

The user should configure the deinterlacer registers for the correct video raster size, basic "field interpolation mode" and then schedule the Deinterlacer to start on the next frame. At the next frame boundary the Deinterlacer will become "synchronised" this can be seen at the top level pin "deint_sync" and also by an interrupt or reading the status register.

At this point the Deinterlacer will now start producing deinterlaced video output. The output video interface pixel rate will now double. If a fault occurs then the Deinterlacer will either lose sync or generate a fifo error. The porbable reasons for these are :

- Loss of sync; due to automatic recovery from an internal fifo overrun error, or the X/Y dimensions do not match the input video X/Y dimensions. The Deinterlacer must internal track X/Y so these registers must match.
- System error; if this is the first time the error has been seen, then the likelihood is either that the vid_clk is not fast enough to allow pixel output at 2x the input rate, or the output fifo has stalled the video and a backlog of >1000 pixels has occurred inside the Deinterlacer.

Step 3: Full Deinterlacing Using Memory Controller

At this point the Deinterlacer should be doing on the fly deinterlacing without the use of the VFBC/AXI-MM port. Program the base addresses of the triple buffers to target a unique area of eternal memory. Update the mode register to select motion/full deinterlacing method. The Deinterlacer will on the next frame boundary start the memory interface port.

Under normal operation only the frame interrupt should ever trigger.

If a system error occurs, they can be broken down into 3 types

- Write stream overflow; the VFBC write port or AXI-Write port does not have enough bandwidth to keep up with the demand off the Deinterlacer. If possible and applicable, either increase the vfbc_clk rate or in the case of AXI, increase the data width of this port.
- Read stream 0, Read stream 1 underflow; Either of the two internal read datapaths fifo's have under run. This is generally due to eccessive system latency, or to slow a vfbc_clk rate.

Possible chipscope analysis using an AXI -Bus-Monitor would best help understand the bottleneck here.

Step 4: Check the Algorithms for Incorrect Video Output

By using the inbuilt colourisation mode, the diagonal and motion kernel operations can be tracked. Turn on the colourisation modes and observe the output video. Using a known video test sequence the colourisation should show the motion aritfiacts and diagonal edge detections (only in moving objects). If the motion trails do not match the image then there is most likely data corruption in the external memory interface port. Although the transactions might be running cleanly, the triple buffers data would seem to be corrupt.

If corruption is visible, by activating PsF mode, the Deinterlacer is forced to use the external memory for every Deinterlaced video line. By enabling this mode, the user can validate the external memory is not corrupt.

Step 5: Pulldown Testing and Pitfalls

When applicable, the inbuilt cadence detectors can be individually enabled / disabled. Once enabled, the detectors will periodically activate/deactivate. In images with low/no motion the cadence detectors may disable until such time as significant motion occurs again. This is normal operation. If the user is monitoring the pulldown interrupts they will see this periodic cycling.

For instance, in the case of scene changes through black, the cadence detector may also drop out momentarily. As no motion is visible at this point the quality of the video output will still be of highest quality even though the cadence detector is inactive.

Failure to detect a cadence in a known sequence that should have 3:2 or 2:2 is generally down to poor quality video that has undergone various compression's/re-authoring steps. For example in converting a DVD to SDI, the quality of the hardware decoders and subsequent scalers, colour-space converters, chroma-resamplers etc.. can all introduce sufficient noise and artifacts that makes the cadence become undetectable. This is specially in the case of 2:2 footage, 3:2 encoding is a more robust mechanism.

XILINX®

Appendix C

Application Software Development

pCore Driver Files

The Deinterlacer pCore includes a software driver written in the C programming language that the user can use to control the core. A high-level API provides application developers easy access to the features of the Xilinx® Object Segmentation core. A low-level API is also provided for developers to access the core directly through the system registers described in the previous section.

Table C-1 lists the files included with the Deinterlacer pCore driver.

Table C-1:

Name	Description
xdeint.h	Contains all prototypes of high-level API to access all of the features of the Xilinx Video Deinterlacer device.
xdeint.c	Contains the implementation of high-level API to access all of the features of the Xilinx Video Deinterlacer device
xdeint_intr.c	Contains the implementation of high-level API to access the interrupt feature of the Xilinx Video Deinterlacer device.
xdeint_sinit.c	Contains static initialization methods for the Xilinx Video Deinterlacer device.
xdeint_g.c	Contains a template for a configuration table of Xilinx Video Deinterlacer devices. This file is used by the high-level API and is automatically generated to match the Xilinx Video Deinterlacer device configuration by Xilinx EDK/SDK tools when the software project is built.
xdeint_hw.h	Contains low-level API (that is, identifiers and register-level driver API) that can be used to

pCore API Functions

This section describes the functions included in the pcore Driver files generated for the Deinterlacer pCore. The software API is provide to allow easy access to the registers of the pCore as defined in Table 2-1. To utilize the API functions provided, the following header files must be included in the user's C code:

```
#include "xparameters.h"

#include "xdeint.h"
```

The hardware settings of your system, including the base address of your Object Segmentation core are defined in the xparameters.h file. The xdeint.h file provides the API access to all of the features of the Deinterlacer device driver. More detailed documentation of the API functions can be found by opening the file index.html in the pCore directory deinterlacer_v1_00_a/doc/html/api

pCore API Functions in xdeint.c

- int XDeint_ConfigInitialize(XDeint *InstancePtr, XDeint_Config *CfgPtr,u32 EffectiveAddr)

  This function initializes an Deinterlacer device
- XDeint_SetFramestore(XDeint *InstancePtr,u32 FieldAddr1, u32 FieldAddr2,u32 FieldAddr3, u32 FrameSize)

  This function initialises the buffer pointers used by the deinterlacers external memory interface
- XDeint_SetVideo(XDeint *InstancePtr, u32 Packing, u32 Colour, u32 Order, u32 PSF)

  This function sets up the type of video processing to perform
- XDeint_SetThresholds(XDeint *InstancePtr u32 t1, u32 t2)

  This function alters the default motion thresholds
- XDeint_SetPulldown(XDeint *InstancePtr u32 lo, u32 hi, u32 enable)

  This function enabled/disables the pulldown/cadence detectors
- void XDeint_GetVersion(XDeint *InstancePtr, u16 *Major, u16 *Minor, u16 *Revision)

  This function returns the hardware version
- void XDeint_SetSize(XDeint *InstancePtr, u32 Width, u32 Height)

  This function sets the video frame aperture of the input video.

Functions in xdeint_sinit.c

- XDeint_Config *XDeint_LookupConfig(u16 DeviceId)

  This function returns a references to an Xdeint_config structure based on the unique device ID given

Functions in xdeint_intr.c

- void XDeint_IntrHandler(void *InstancePtr)

  This function is the interrupt handler for the Deinterlacer driver
- int XDeint_SetCallBack(XDeint *InstancePtr,void *CallBackFunc)

  This function installs an asynchronous callback function for the given handler type.

XILINX®

Appendix D

C Model Reference

The Xilinx LogiCORE™ IP Video Deinterlacer has a bit accurate C model for 32-bit Windows, 64-bit Windows, 32-bit Linux and 64-bit Linux platforms. The model has an interface consisting of a set of C functions, which reside in a statically link library (shared library). Full details of the interface are given in Interface. An example piece of C code is provided in Example Code to show how to call the model. The model is bit accurate, as it produces exactly the same output data as the core on a frame-by-frame basis. However, the model is not cycle accurate, as it does not model the core's latency or its interface signals. The latest version of the model is available for download on the Xilinx LogiCORE IP Video Deinterlacer web page at:

http://www.xilinx.com/products/ipcenter/EF-DI-DEINTERLACER.htm

Unpacking and Model Contents

Unzip the `deinterlacer_v1_0_bitacc_model.zip` file, containing the bit accurate models for the Video Deinterlacer IP Core. This creates the directory structure and files in Table D-1.

Table D-1: **Directory Structure and Files of the Video Deinterlacer v1.0 Bit Accurate C Model**

File Name	Contents
./doc	Documentation directory
README.txt	Release notes
pg017_deinterlacer.pdf	LogiCORE IP Video Deinterlacer Product Guide
Makefile	Makefile for running gcc via make for 32-bit and 64-bit Linux platforms
deinterlacer_v1_0_bitacc_cmodel.h	Model header file
yuv_utils.h	header file declaring the YUV image / video container type and support functions including .yuv file I/O
rgb_utils.h	header file declaring the RGB image / video container type and support functions
bmp_utils.h	header file declaring the bitmap (.bmp) image file I/O functions.
video_utils.h	header file declaring the generalized image / video container type, I/O and support functions
video_fio.h	header file declaring support functions for testbench stimulus file I/O
run_bitacc_cmodel.c	example code calling the C model

Table D-1: **Directory Structure and Files of the Video Deinterlacer v1.0 Bit Accurate C Model**

File Name	Contents
./lin64	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms.
libIp_deinterlacer_v1_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	Xilinx STL library, referenced by libIp_deinterlacer_v1_0_bitacc_cmodel.so
run_bitacc_cmodel	64-bit Linux fixed configuration executable
run_bitacc_cmodel_config	64-bit Linux programmable configuration executable
./lin	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms.
libIp_deinterlacer_v1_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	Xilinx STL library, referenced by libIp_deinterlacer_v1_0_bitacc_cmodel.so
run_bitacc_cmodel	32-bit Linux fixed configuration executable
run_bitacc_cmodel_config	32-bit Linux programmable configuration executable
./nt64	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms.
libIp_deinterlacer_v1_0_bitacc_cmodel.dll	Precompiled dynamic link library file for 64-bit Windows platforms compilation
libIp_deinterlacer_v1_0_bitacc_cmodel.lib	Precompiled static library file for 64-bit Windows platforms compilation
stlport.5.1.dll	Xilinx STL library
run_bitacc_cmodel.exe	64-bit Windows fixed configuration executable
run_bitacc_cmodel_config.exe	64-bit Windows programmable configuration executable
./nt	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms.
libIp_deinterlacer_v1_0_bitacc_cmodel.dll	Precompiled dynamic link library file for 32-bit Windows platforms compilation
libIp_deinterlacer_v1_0_bitacc_cmodel.lib	Precompiled static library file for 32-bit Windows platforms compilation
stlport.5.1.dll	Xilinx STL library
run_bitacc_cmodel.exe	32-bit Windows fixed configuration executable
run_bitacc_cmodel_config.exe	32-bit Windows programmable configuration executable
./examples	Example input files to be used with the run_bitacc_cmodel executable
FormulaOne_035.yuv	Example YUV input file
FormulaOne_036.yuv	Example YUV input file
FormulaOne_037.yuv	Example YUV input file

Table D-1: **Directory Structure and Files of the Video Deinterlacer v1.0 Bit Accurate C Model**

File Name	Contents
FormulaOne_038.yuv	Example YUV input file
FormulaOne_039.yuv	Example YUV input file
FormulaOne_040.yuv	Example YUV input file
FormulaOne_041.yuv	Example YUV input file
FormulaOne_042.yuv	Example YUV input file

Installation

For Linux, make sure the following files are in a directory in the `$LD_LIBRARY_PATH` environment variable:

- `libIp_deinterlacer_v1_0_bitacc_cmodel.so`
- `libstlport.so.5.1`

Software Requirements

The Video Deinterlacer C models were compiled and tested with the software listed in Table D-2.

Table D-2: **Compilation Tools for the Bit Accurate C Models**

Platform	C Compiler
64-bit Linux	GCC 3.4.6 & 4.1.1
32-bit Linux	GCC 3.4.6 & 4.1.1
64-bit Windows	Microsoft Visual Studio 2008
32-bit Windows	Microsoft Visual Studio 2008

Interface

The Xilinx LogiCORE IP Video Deinterlacer bit-accurate C model core function is provided as a statically linked library. The bit-accurate C model is accessed through a set of functions and data structures, declared in the header file `deinterlacer_v1_0_bitacc_cmodel.h`. A higher-level software project may make function-calls to the functions below:

```
/**
 * Create a new state structure for this C-Model.
 *
 * IMPORTANT: Client is responsible for calling
 *            xilinx_ip_deinterlacer_v1_0_destroy_state()
 *            to free state memory.
 *
 * @param generics    Generics to be used to configure C-Model
 *                    state.
 *
 * @returns xilinx_ip_deinterlacer_v1_0_state*  Pointer to the internal
 *                                              state.
```

```
 */
struct xilinx_ip_deinterlacer_v1_0_state*
xilinx_ip_deinterlacer_v1_0_create_state(struct
xilinx_ip_deinterlacer_v1_0_generics generics);

/**
 * Simulate this bit-accurate C-Model.
 *
 * @param     state      Internal state of this C-Model. State
 *                       may span multiple simulations.
 * @param     inputs     Inputs to this C-Model.
 * @param     outputs    Outputs from this C-Model.
 *
 * @returns   Exit code   Zero for SUCCESS, Non-zero otherwise.
 */
int xilinx_ip_deinterlacer_v1_0_bitacc_simulate
(
 struct xilinx_ip_deinterlacer_v1_0_state*   state,
 struct xilinx_ip_deinterlacer_v1_0_inputs   inputs,
 struct xilinx_ip_deinterlacer_v1_0_outputs* outputs
);
```

Before using the model, the structures holding the generics, inputs, and outputs of the Deinterlacer instance have to be defined:

```
struct  xilinx_ip_deinterlacer_v1_0_generics generics;
struct  xilinx_ip_deinterlacer_v1_0_inputs   inputs;
struct  xilinx_ip_deinterlacer_v1_0_outputs  outputs;
```

Declaration of the above structures can be found in `deinterlacer_v1_0_bitacc_cmodel.h`.

Before making the function calls, the following steps are necessary:

1. Populate the 'generics' structure. It defines the values of build-time parameters. Please see Deinterlacer Generics Structure for more information on the structure and an example of how to initialize.
2. Populate the 'inputs' structure. It defines the values of run-time parameters. Please see Deinterlacer Inputs Structure for more information on the structure and an example of how to initialize.
3. Populate the 'outputs' structure. Please see Deinterlacer Outputs Structure for more information on the structure and an example of how to initialize.

After the inputs are defined and all `video_structs` initialized the model can be simulated by calling the following functions

```
  state = xilinx_ip_deinterlacer_v1_0_create_state(generics);
  if (state == NULL) {
    printf("ERROR: could not create state object\n");
    return 1;
  }

  // Simulate the core
  printf("Running the C model...\n");
  if(xilinx_ip_deinterlacer_v1_0_bitacc_simulate(state, inputs,
&outputs) != 0) {
    printf("ERROR: simulation did not complete successfully\n");
    return 1;
  } else {
    printf("Simulation completed successfully\n");
```

```
}
```

Results are provided in the outputs structure, which contains only one member of type `video_struct`. More information on the video_struct structure can be found in Deinterlacer Video Structure. Successful execution of all provided functions return value 0, otherwise a non-zero error code indicates that problems were encountered during function calls.

Deinterlacer Generics Structure

The Xilinx LogiCORE IP Video Deinterlacer Core bit accurate C model takes multiple generic parameters. All generic parameters are integers or integer arrays. See Table D-3 for generic definitions.

Table D-3: **Deinterlacer Generics Structure**

Generic	Designation
C_STREAMS	Number of simultaneous color planes Valid values are 2 or 3.
C_DEPTH	Bit depth of a pixel Valid values are 8, 10 or 12
C_DIAG	Enable the diagonal kernel 0 = disables the diagonal kernel 1 = enables the diagonal kernel
C_MOTION	Enable the motion kernel 0 = disables the motion kernel 1 = enables the motion kernel
C_PULLDOWN	Cadence/Pull-down detection 0 = No pull-down detection 1 = Full pull-down detection
C_COL	Static color space setting 0 = YUV 1 = RGB

Calling `xilinx_ip_deinterlacer_v1_0_get_default_generics()` initializes the generics structure, `xilinx_ip_deinterlacer_v1_0_generics`, with the deinterlacer defaults. An example of initialization of the generics structure is as follows:

```
generics = xilinx_ip_deinterlacer_v1_0_get_default_generics(); //Get
Defaults
```

Deinterlacer Inputs Structure

The structure `xilinx_ip_deinterlacer_v1_0_inputs` defines the values of run time parameters and the actual input video frames/images.

```
struct xilinx_ip_deinterlacer_v1_0_inputs
{
  struct  video_struct video_in;
```

```
  struct  deinterlacer_cfg_struct *cfg;
  struct  deinterlacer_pull_struct *pull;


}; // end xilinx_ip_deinterlacer_v1_0_inputs
```

The `video_in` variable is an array of video_struct structures, one structure per layer. See the Deinterlacer Video Structure for a description of the `video_in` structure. The `video_in` structure must be initialized.

Deinterlacer Config Structure

The `cfg` variable is a pointer to the `deinterlacer_cfg_struct`. The `deinterlacer_cfg_struct` is defined as:

```
struct deinterlacer_cfg_struct
{
  int frame;
  int bmpfiles;
  int txtfiles;
  int rate;
  int t1;
  int t2;
  int pull_lo;
  int pull_hi;
  int pixel_scale;
  int filewidth;
  int fileheight;
  int depth;
  int format;
  int mode;
  int order;
  int pulldown;
  int cropx;
  int cropy;
  int width;
  int height;
  int length;
  int index;
  int debug;
  int pixel_mask;
  char source[256];
  char prefix[256];
  char num_len;
  char suffix[256];
  char golden[256];
  FILE *avifile;

  int lut[4096];

};
```

Pulldown Structure

The pull variable is a pointer to the `deinterlacer_pull_struct`. The `deinterlacer_pull_struct` is defined as:

```
struct deinterlacer_pull_struct{
  // Internal 22 State Machine
  int trained_22;
  int trained_22_d1;
  int last_22_delta;
  int confidence_22;

  // Internacer 32 State Machine
  int cx_32;
  int switch_32;
  int next_field_32;
  int bad_time_32;
  int bad_32;
  int trained_32;
  int trained_32_d1;
  int state_32;
  int p24_32;

  // Top level cotrol
  int active_32_early;
  int active_32;
  int active_22_early;
  int active_22;
  int mux_switch;
  int next_field;
  int p24;
};
```

Deinterlacer Outputs Structure

The structure `xilinx_ip_deinterlacer_v1_0_outputs` provides the actual output video frames/images of the deinterlacer core. This structure is a wrapper to the standard `video_struct` used by other Xilinx video core C models.

```
struct xilinx_ip_ deinterlacer_v1_0_outputs
{
struct video_struct video_out;
}; // xilinx_ip_deinterlacer_v1_0_outputs
The video_out structure must be initialized. The following code shows a
typical
video_out initialization.
// Setup Output Video Buffer
outputs.video_out.frames = inputs.num_frames;
outputs.video_out.rows = inputs.frame_cfg->y_size;
outputs.video_out.cols = inputs.frame_cfg->x_size;
outputs.video_out.mode = FORMAT_C444;
outputs.video_out.bits_per_component = generics.C_DATA_WIDTH;
outputs.video_out.data[0] = NULL;
outputs.video_out.data[1] = NULL;
outputs.video_out.data[2] = NULL;
```

Deinterlacer Video Structure

Input images or video streams can be provided to the Deinterlacer v1.0 reference model using the `video_struct` structure, defined in `video_utils.h`. Output images or video streams are also placed within a `video_struct` structure. The `video_struct` is defined as:

```
struct video_struct{
  int frames, rows, cols, bits_per_component, mode;
  uint16*** data[5];
};
```

The structure member variables are defined in Table D-4.

Table D-4: **Member Variables of the Video Structure**

Member Variable	Designation
frames	Number of video/image frames in the data structure
rows	Number of rows per frame Pertains to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through the all frames of the video stream, however different planes, such as y, u and v can have different smaller dimensions.
cols	Number of columns per frame Pertains to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through the all frames of the video stream, however different planes, such as y, u and v can have different smaller dimensions.
bits_per_component	Number of bits per color channel/component. All image planes are assumed to have the same color/ component representation. Maximum number of bits per component is 16.
mode	Contains information about the designation of data planes. Named constants to be assigned to mode are listed in Table D-5.
data	Of 5 pointers to 3 dimensional arrays containing data for image planes. data is in 16 bit unsigned integer format accessed as data[plane][frame][row][col]

Table D-5 shows the named constants for video modes with corresponding planes and representations.

Table D-5: **Named Constants for Video Modes**

Mode	Planes	Video Representation
FORMAT_MONO	1	Monochrome - Luminance only.
FORMAT_RGB	3	RGB image / video data
FORMAT_C444	3	444 YUV, or YCrCb image / video data
FORMAT_C422	3	422 format YUV video, (u,v chrominance channels horizontally sub-sampled)
FORMAT_C420	3	420 format YUV video, (u,v sub-sampled both horizontally and vertically)
FORMAT_MONO_M	3	Monochrome (Luminance) video with Motion.

Table D-5: **Named Constants for Video Modes *(Cont'd)***

Mode	Planes	Video Representation
FORMAT_RGBA	4	RGB image / video data with alpha (transparency) channel
FORMAT_C420_M	5	420 YUV video with Motion or Alpha
FORMAT_C422_M	5	422 YUV video with Motion or Alpha
FORMAT_C444_M	5	444 YUV video with Motion or Alpha
FORMAT_RGBM	5	RGB video with Motion

Working With Video_struct Containers

The `video_utils.h` file defines functions to simplify access to video data in `video_struct`.

```
int video_planes_per_mode(int mode);
int video_rows_per_plane(struct video_struct* video, int plane);
int video_cols_per_plane(struct video_struct* video, int plane);
```

Function video_planes_per_mode returns the number of component planes defined by the mode variable, as described in Table D-5. Functions `video_rows_per_plane` and `video_cols_per_plane` return the number of rows and columns in a given plane of the selected video structure. The following example demonstrates using these functions in conjunction to process all pixels within a video stream stored in variable `in_video`, with this construct:

```
for (int frame = 0; frame < in_video->frames; frame++) {
  for (int plane = 0; plane < video_planes_per_mode(in_video->mode);
plane++) {
    for (int row = 0; row < rows_per_plane(in_video,plane); row++) {
      for (int col = 0; col < cols_per_plane(in_video,plane); col++) {
        // User defined pixel operations on
        // in_video->data[plane][frame][row][col]
      }
    }
  }
}
```

Delete the Video Structure

Large arrays such as the video_in element in the video structure must be deleted to free up memory. As an example, the following function is defined as part of the `video_utils` package.

```
void free_video_buff(struct video_struct* video )
{
  int plane, frame, row;

  if (video->data[0] != NULL) {
   for (plane = 0; plane <video_planes_per_mode(video->mode); plane++)
{
      for (frame = 0; frame < video->frames; frame++) {
        for (row = 0; row<video_rows_per_plane(video,plane); row++) {
          free(video->data[plane][frame][row]);
        }
```

```
            free(video->data[plane][frame]);
          }
          free(video->data[plane]);
        }
      }
    }
```

This function can be called in the following way to free the video input buffers (up to eight) and the video output buffer:

```
// Free Layer Buffers
for(i=0; i < generics.C_NUM_LAYERS; i++)
{
printf("Freeing Layer Video Buffer #%d...\n", i);
free_video_buff(&inputs.video_in[i]);
}
printf("Freeing Output Buffer...\n");
free_video_buff(&outputs.video_out);
```

Example Code

Two example C files, `run_bitacc_cmodel.c` and `run_bitacc_cmodel_config.c`, are provided. The 32-bit and 64-bit Windows and Linux executables for these examples are also included.

The `run_bitacc_cmodel` example executable provides:

- Shows a fixed implementation of the Deinterlacer
- Contains an example of how to write an application that makes all necessary function calls to the Deinterlacer C model core function.
- Contains an example of how to populate the video structures at the input and output, including allocation of memory to these structures.
- Uses a YUV file reading function to extract video information from YUV files for use by the model.
- Uses a YUV file writing function to provide an output YUV file, which allows the user to visualize the result of the core.

The `run_bitacc_cmodel` example executable does not use command line parameters. To run the executable:

1. Use the cd command to go to the platform directory (lin64, lin, win64 or win32).
2. Enter this command at the shell or DOS prompt:

   ```
   run_bitacc_cmodel
   ```

The `run_bitacc_cmodel_config` example executable provides:

- Shows configurable implementations of the Deinterlacer configured from command line arguments.
- Includes a command line parser, allowing the user to pass parameters into the model for multiple test cases.
- Uses YUV or BMP file reading functions to extract video information from YUV or BMP files for use by the model.
- Uses YUV or BMP file writing functions to provide an output YUV or BMP file, which allows the user to visualize the result of the core.

The `run_bitacc_cmodel_config` example executable uses multiple command line parameters. To run the executable:

1. Use the cd command to go to the platform directory (lin64, lin, win64 or win32).
2. Enter this command at the shell or DOS prompt:

   ```
   ./run_bitacc_cmodel_config <-parameter> <value> …
   ```

   For example:

   ```
   run_bitacc_cmodel_config -width 720 -height 576 -depth 8 -mode full
   -length 2 -source test_000.yuv
   ```

Command Line Options in Detail

The following is a detailed list of the options:

- **-core**: selects which gate-level model is run; excluding this option defaults to RTL simulation.
- **-format**: selects the input file format; possible input formats are 422YUV8, 422YUV10, 444BMP.
- **-rate**: selects output AVI files frame rate.
- **-order**: selects which field order is used to store the source files. By choosing "pal", line 1 is temporally used before line 2. By choosing NTSC, this order is reversed,
- **-pulldown**: selects the operation of the pulldown detector; it can be either switched on or off.
- **-mode**: selects what internal processing is used to generate a deinterlaced image. If "none" is selected, the output is field interpolated. If "motion" is selected, then only the motion adaptive algorithm is used. If "diag" is selected, then only the diagonal algorithm is used. If "full", then all features are enabled.
- **-cropx, -cropy, -cropxsize, -cropysize**: allow for a region of interest to be extracted from a given source image; the origin of a picture is assumed to be 0,0 and only even x offsets are allowed.
- **-width**: sets the full pixel width of the input file image and is required.
- **-height**: sets the full pixel height of the input file image and is required.
- **-length**: sets the number of files read by the chosen core; it should be set greater than three to allow enough priming of the motion adaptive datapath.
- **-txt**: used by the C model to generate a .txt equivalent file set of the source images, which are then used by the VHDL or Verilog models.
- **-source**: path and file name of the first file to be read.
- **-debug**: enables colorized images to be generated.

Initializing the Deinterlacer Input Video Structure

The easiest way to assign stimuli values to the input video structure is to initialize it with an image or video. The `bmp_util.h`, `yuv_utils.h`, `rgb_utils.h` and `video_util.h` header files packaged with the bit accurate C models contain functions to facilitate file I/O.

Bitmap Image Files

The rgb_utils.h and bmp_utils.h files declare functions that help access files in Windows bitmap format (http://en.wikipedia.org/wiki/BMP_file_format). However, this format

limits color depth to a maximum of 8 bits per pixel, and operates on images with three planes (R,G,B). Consequently, the following functions operate on arguments type rgb8_video_struct, which is defined in rgb_utils.h. Also, both functions support only true color, non-indexed formats with 24 bits per pixel.

```
int write_bmp(FILE *outfile, struct rgb8_video_struct *rgb8_video);
int read_bmp(FILE *infile, struct rgb8_video_struct *rgb8_video);
```

These functions are used to dynamically allocate and free memory for RGB structure storage:

```
int alloc_rgb8_frame_buff(struct rgb8_video_struct* rgb8video );
void free_rgb_frame_buff(struct rgb_video_struct* rgb_video );
```

Exchanging data between rgb8_video_struct and general video_struct type frames/videos is facilitated by functions:

```
int copy_rgb8_to_video(struct rgb8_video_struct* rgb8_in,
struct video_struct* video_out );
int copy_video_to_rgb8( struct video_struct* video_in,
struct rgb8_video_struct* rgb8_out );
```

Note: All image / video manipulation utility functions expect both input and output structures initialized; for example, pointing to a structure that has been allocated in memory, either as static or dynamic variables. Additionally, the input structure must have the dynamically allocated containers (data, r, g, b, y, u, and v arrays) already allocated and initialized with the input frame(s). If the output container structure is pre-allocated at the time of the function call, the utility functions verify and issue an error if the output container size does not match the size of the expected output. If the output container structure is not pre-allocated, the utility functions create the appropriate container to hold results.

YUV Image/Video Files

The yuv_utils.h file declares functions that support file access in YUV format. These functions are used to dynamically allocate and free memory for YUV structure storage:

```
int alloc_yuv8_frame_buff(struct yuv8_video_struct* yuv8video );
void free_yuv_frame_buff(struct yuv_video_struct* yuv_video );
```

These functions allow reading and writing of YUV functions (used to initialize or write yuv8_video data):

```
int write_yuv(FILE *outfile, struct yuv8_video_struct *yuv8_video);
int read_yuv(FILE *infile, struct yuv8_video_struct *yuv8_video);
```

Exchanging data between yuv8_video_struct and general video_struct type frames/videos is facilitated by functions:

```
int copy_yuv8_to_video(struct yuv8_video_struct* yuv8_in,
struct video_struct* video_out );
int copy_video_to_yuv8( struct video_struct* video_in,
struct yuv8_video_struct* yuv8_out );
```

YUV formats (4:2:0, 4:2:2 and 4:4:4) can be converted with these functions:

```
int yuv8_420to444(struct yuv8_video_struct* video_in, struct
yuv8_video_struct* video_out);
int yuv8_422to444(struct yuv8_video_struct* video_in, struct
yuv8_video_struct* video_out);
int yuv8_444to420(struct yuv8_video_struct* video_in, struct
yuv8_video_struct* video_out);
int yuv8_444to422(struct yuv8_video_struct* video_in, struct
yuv8_video_struct* video_out);
```

Binary Image/Video Files

The video_utils.h file declares functions that help load and save generalized video files in raw, uncompressed format. These functions effectively serialize the video_struct structure:

```
int read_video( FILE* infile, struct video_struct* in_video);
int write_video(FILE* outfile, struct video_struct* out_video);
```

The corresponding file contains a small, plain text header defining, "Mode", "Frames", "Rows", "Columns", and "Bits per Pixel". The plain text header is followed by binary data, 16-bits per component in scan line continuous format. Subsequent frames contain as many component planes as defined by the video mode value selected. Also, the size (rows, columns) of component planes can differ within each frame as defined by the actual video mode selected.

These functions are used to dynamically allocate and free memory for video structure storage:

```
int alloc_video_buff(struct video_struct* video );
void free_video_buff(struct video_struct* video );
```

Compiling on 32-bit and 64-bit Windows Platforms

Precompiled library deinterlacer_v1_0_bitacc_cmodel.lib, top level demonstration code run_bitacc_cmodel_config.c and example code run_bitacc_cmodel.c must be compiled with an ANSI C compliant compiler under Windows 32-bit or Windows 64-bit. This section describes an example using Microsoft Visual Studio. In Visual Studio create a new, empty Win32 Console Application project. As existing items, add:

- `libIpdeinterlacer_v1_0_bitacc_cmodel.lib` to the "Resource Files" folder of the project
- `run_bitacc_cmodel.c or the run_bitacc_cmodel_config.c` to the "Source Files" folder of the project
- `deinterlacer_v1_0_bitacc_cmodel.h header` file to the "Header Files" folder of the project
- `bmp_utils.h` file to the "Header Files" folder of the project
- `rgb_utils.h` file to the "Header Files" folder of the project
- `video_fio.h` file to the "Header Files" folder of the project
- `video_utils.h` file to the "Header Files" folder of the project
- `yuv_utils.h` file to the "Header Files" folder of the project

To build the x64 executable for 64-bit Windows platforms, perform these steps. These steps can be skipped if building the Win32 executable.

1. Right-click on the solution in the Solution Explorer and click Properties at the bottom of the pop-up menu.
2. Click **Configuration Manager**.
3. In the Active solution platform drop-down box, select <New…>.
4. In the new platform drop-down box, select x64 and click OK.
   Make sure that all the projects now have x64 as the default platform in the Configuration Manager.
5. After the project is created and populated, it must be compiled and linked (built) to create a Win32 or x64 executable. To perform the build step, select Build Solution from the Build menu. An executable matching the project name is created either in the

Debug or Release subdirectories under the project location based on whether "Debug" or "Release" has been selected in the "Configuration Manager" under the Build menu.

Note: The run_bitacc_cmodel.c file is an example demonstration that reads no input but generates an output .yuv file from internally generated test patterns. The run_bitacc_cmodel_config.c file is a configurable demonstration and requires several input files to run. See Running the Executables for information on command line arguments and input file formats.

Compiling under 32-bit and 64-bit Linux Platforms

Example Demonstration

To compile the example demonstration, go to the directory where the header files, the library files and `run_bitacc_cmodel.c` were unpacked. The libraries and header files are referenced during the compilation and linking process. In this directory, perform these steps:

1. Set your LD_LIBRARY_PATH environment variable to include the root directory where the model zip file was unzipped. For example:

```
setenv LD_LIBRARY_PATH <unzipped_c_model_dir>:${LD_LIBRARY_PATH}
```

2. Copy these files from the /lin32 or /lin64 directory to the root directory:

```
libstlport.so.5.1
libIp_deinterlacer_v1_0_bitacc_cmodel.so
```

3. In the root directory, compile using the GNU C Compiler by typing this command at the shell prompt:

```
gcc -m32 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
run_bitacc_cmodel -L. -lIp_deinterlacer_v1_0_bitacc_cmodel
-Wl,-rpath,.
gcc -m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
run_bitacc_cmodel -L. -lIp_deinterlacer_v1_0_bitacc_cmodel
-Wl,-rpath,.
```

4. This results in the creation of the executable `run_bitacc_cmodel`, which can be run using this command:

```
./run_bitacc_cmodel
```

A make file is also included that runs GCC. To clean the executable and compile the example code, enter this command at the shell prompt:

```
make clean all
```

Configurable Demonstration

To compile the configurable demonstration, go to the directory where the header files, the library files and `run_bitacc_cmodel_config.c` were unpacked. The libraries and header files are referenced during the compilation and linking process. In this directory perform these steps:

1. Set your LD_LIBRARY_PATH environment variable to include the root directory where the model zip-file was unzipped. For example:

```
setenv LD_LIBRARY_PATH <unzipped_c_model_dir>:${LD_LIBRARY_PATH}
```

2. Copy these files from the /lin64 directory to the root directory:

```
libstlport.so.5.1
```

```
libIp_deinterlacer_v1_0_bitacc_cmodel.so
```

3. In the root directory, compile using the GNU C Compiler by entering this command at the shell prompt:

   ```
   gcc -x c++ run_bitacc_cmodel_config.c -o run_bitacc_cmodel_config -L.
   -lIp_deinterlacer_v1_0_bitacc_cmodel -Wl,-rpath,.
   ```

4. This results in the creation of the executable run_bitacc_cmodel, which can be run using this command:

   ```
   ./run_bitacc_cmodel_config <-parameter> <value> …
   ```

   For example:

   ```
   run_bitacc_cmodel_config -width 720 -height 576 -depth 8 -mode full
   -length 2 -source test_000.yuv
   ```

   A make file is also included that runs GCC. To clean the executable and compile the example code, enter this following command at the shell prompt:

   ```
   make clean run_bitacc_cmodel_config
   ```

Running the Executables

Included in the zip file are precompiled executable files for use with 32-bit and 64-bit Windows and Linux platforms. The instructions for running on each platform are included in this section.

Example Demonstration

The example demonstration does not use command line parameters. To run on a 32-bit or 64-bit Linux platform, perform these steps:

1. Set your $LD_LIBRARY_PATH environment variable to include the root directory where the model zip file was unzipped. For example:

   ```
   setenv LD_LIBRARY_PATH <unzipped_c_model_dir>:${LD_LIBRARY_PATH}
   ```

2. Copy these files from the /lin64 (for 64-bit Linux) or from the /lin (for 32-bit Linux) directory to the root directory:

   ```
   libstlport.so.5.1
   libIp_deinterlacer_v1_0_bitacc_cmodel.so
   run_bitacc_cmodel
   ```

3. Execute the model. From the root directory, enter this command at a shell prompt:

   ```
   run_bitacc_cmodel
   ```

To run on a 32-bit or 64-bit Windows platform, perform these steps:

1. Copy this file from the /nt64 (for 64-bit Windows) or from the /nt (for 32-bit Windows) directory to the root directory:

   ```
   run_bitacc_cmodel.exe
   ```

2. Execute the model. From the root directory, enter this command at a DOS prompt:

   ```
   run_bitacc_cmodel
   ```

During successful execution, the `c_deint0000.bmp` file is created in the directory containing the `run_bitacc_cmodel` executable. This file bitmap file. The example demonstration is set up to generate 15 frames of video data at 200x120 24-bit format.

Configurable Demonstration

The configurable demonstration takes multiple command line parameters. To run on a 32-bit or 64-bit Linux platform, perform these steps:

1. Set your $LD_LIBRARY_PATH environment variable to include the root directory where the model zip-file was unzipped. For example:

   ```
   setenv LD_LIBRARY_PATH <unzipped_c_model_dir>:${LD_LIBRARY_PATH}
   ```

2. Copy these files from the /lin64 (for 64-bit Linux) or from the /lin (for 32-bit Linux) directory to the root directory:

   ```
   libstlport.so.5.1
   libIp_deinterlacer_v1_0_bitacc_cmodel.so
   run_bitacc_cmodel_config
   ```

3. Execute the model. From the root directory, enter this command at a shell prompt:

   ```
   ./run_bitacc_cmodel_config <-parameter> <value> …
   ```

   For example:

   ```
   run_bitacc_cmodel_config -width 720 -height 576 -depth 8 -mode full
   -length 2 -source test_000.yuv
   ```

To run on a 32-bit or 64-bit Windows platform, perform these steps:

1. Copy this file from the /nt64 (for 64-bit Windows) or from the /nt (for 32-bit Windows) directory to the root directory:

   ```
   run_bitacc_cmodel_config.exe
   ```

2. Execute the model. From the root directory, enter this command at a DOS prompt:

   ```
   ./run_bitacc_cmodel_config <-parameter> <value> …
   ```

   For example:

   ```
   run_bitacc_cmodel_config -width 720 -height 576 -depth 8 -mode full
   -length 2 -source test_000.yuv
   ```

During successful execution, multiple bitmap files are created in the directory containing the `run_bitacc_cmodel_config` executable.

Each individual simulation is invoked using a binary executable script and some command line parameters. The main parameters are used to steer the test and its target. The options for a test are shown in Table D-6.

Table D-6: **Simulation Options**

Option Name	Description	Option Values	Default
depth	Bit depth of video stream	8 | 10 | 12	10
format	File format used	yuv8|yuv10|bmp|	yuv8
packing	Pixel packing structure	444 | 422 | 420	444
pulldown	Cadence detector	off | on	off
mode	Deinterlacing type	full | none | motion | diag	full
cropx	Cropping Top Left X	<numeric value>	0
cropy	Cropping Top Left Y	<numeric value>	0
cropxsize	Cropping X size	<numeric value>	<default to width>
cropysize	Cropping Y size	<numeric value>	<default to height>

Table D-6: **Simulation Options**

width	Input File Pixel width	<numeric value>	<error if missing>
height	Input File Pixel height	<numeric value>	<error if missing>
length	Number of files in sequence	<numeric value>	<error if missing>
source	Sequence filename	<filename>	<error if missing>
golden	Sequence filename	<filename>	<used only by compare>
debug	Generate debug images	<numeric value>	0

The "source", "length", "width" and "height" parameters are mandatory, all other missing fields are set to their default.

The following command line shows how to run a C model based, 8-bit full deinterlacer on sequence files "test000":

```
  run_bitacc_cmodel_config -width 720 -height 576 -depth 8 -mode full -length 2 -source
test_000.yuv
```

When running the C model output, two additional AVI files are generated. The first is an animated version of the full deinterlaced sequence, and the second is a side-by-side comparison movie of the motion adaptive and full deinterlacer in operation. This second AVI file allows for easy visual comparison of the outputs. The user can preset the AVI frame rate on the command line.

Simulation Options in Detail

The following is a detailed list of the options:

- **-core**: selects which gate-level model is run; excluding this option defaults to RTL simulation.
- **-format**: selects the input file format; possible input formats are 422YUV8, 422YUV10, 444BMP.
- **-rate**: selects output AVI files frame rate.
- **-order**: selects which field order is used to store the source files. By choosing "pal", line 1 is temporally used before line 2. By choosing NTSC, this order is reversed,
- **-pulldown**: selects the operation of the pulldown detector; it can be either switched on or off.
- **-mode**: selects what internal processing is used to generate a deinterlaced image. If "none" is selected, the output is field interpolated. If "motion" is selected, then only the motion adaptive algorithm is used. If "diag" is selected, then only the diagonal algorithm is used. If "full", then all features are enabled.
- **-cropx**, **-cropy**, **-cropxsize**, **-cropysize**: allow for a region of interest to be extracted from a given source image; the origin of a picture is assumed to be 0,0 and only even x offsets are allowed.
- **-width**: sets the full pixel width of the input file image and is required.
- **-height**: sets the full pixel height of the input file image and is required.
- **-length**: sets the number of files read by the chosen core; it should be set greater than three to allow enough priming of the motion adaptive datapath.
- **-txt**: used by the C model to generate a .txt equivalent file set of the source images, which are then used by the VHDL or Verilog models.

- **-source**: path and file name of the first file to be read.
- **-debug**: enables colorized images to be generated.

XILINX®

Appendix E

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

http://www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

These documents provide supplemental material useful with this user guide:

- UG761, AXI Reference Guide
- DS768, AXI Interconnect IP Data Sheet

Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the IP Release Notes Guide (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Ordering Information

The Video Deinterlacer core is provided under the Xilinx Core License Agreement and can be generated using the Xilinx® CORE Generator™ system. The CORE Generator system is shipped with Xilinx ISE® Design Suite software.

A simulation evaluation license for the core is shipped with the CORE Generator system. To access the full functionality of the core, including FPGA bitstream generation, a full license must be obtained from Xilinx. For more information, visit the Video Deinterlacer product page.

Contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx IP Center.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release.

Notice of Disclaimer