LogiCORE IP Motion Adaptive Noise Reduction v3.0

Product Guide

PG006 October 19, 2011

XILINX®

Table of Contents

LogiCORE IP Motion Adaptive Noise Reduction v3.0

Introduction

The Xilinx LogiCORE™ IP Motion Adaptive Noise Reduction (MANR) is a module for both motion detection and motion adaptive noise reduction in video systems. The core allows the motion detection function to be used independently of the noise reduction function for applications where noise reduction is not needed. The noise reduction algorithm is implemented as a recursive temporal filter with a user programmable transfer function allowing the user to control both the shape of the motion transfer and the strength of the noise reduction applied.

The motion transfer function is initialized according to the settings in the CORE Generator™ GUI, but is also programmable at runtime via the register interface. The CORE Generator™ tool generates the core as either an AXI EDK pCore or as a standalone netlist for a General Purpose Processor (GPP). When generated as an EDK pCore, the processor interface is AXI4-Lite compliant.

Features

- Programmable register control.
- Selectable processor interface:
  - EDK pCore - AXI interface is based on AXI4-Lite specification
  - General Purpose Processor
- Selectable and programmable motion transfer function.
- Full support for interrupts and status registers for easy system control.
- Supports YUV 4:2:2, 4:2:0 at 8 bits per pixel.
- Gives calculated Y/C motion data output for optional use by downstream IP.
- Supports HD frame sizes including 1280x720, and 1920x1080.
- Supports 1080p60, 1080p30, 720p60 and smaller formats.

Applications

- Video Surveillance
- Industrial Imaging
- Video Conferencing
- Machine Vision

LogiCORE IP Facts Table						
Core Specifics						
Supported Device Family(1)	Virtex-7, Kintex-7, Virtex-6, Spartan-6					
Supported User Interfaces	General Purpose Processor (GPP), EDK pCore AXI4-Lite					
	Resources				**Frequency**	
Configuration	**LUTs**	**FFs**	**DSP Slices**	**Block RAMs**	**Max. Freq.**	
	See Table 1-1 through Table 1-4.					
Provided with Core						
Design Files	Netlist for GPP Interfaces, Encrypted Source Code for EDK pCore					
Example Design	Not Provided					
Test Bench	VHDL(2)					
Constraints File	Not Provided					
Simulation Model	VHDL/Verilog Structural Models Bit-accurate C Model(2)					
Tested Design Tools						
Design Entry Tools	CORE Generator, Platform Studio (XPS)					
Simulation(3)	Mentor Graphics ModelSim, Xilinx ISim					
Synthesis Tools	Xilinx Synthesis Technology (XST) 13.3					
Support						
Provided by Xilinx, Inc.						

1. For a complete listing of supported devices, see the release notes for this core.
2. Test bench and C model available on the MANR product page.
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

XILINX®

Chapter 1

Overview

Noise reduction is a common function in video systems and can be used to clean up sensor artifacts or other types of noise present in most video systems. In addition, many surveillance systems and other analytical video processing systems need real-time motion information to provide intelligent processing such as object detection and tracking or camera tampering detection. The MANR core provides both of these capabilities in a single, efficient implementation.

Noise reduction is achieved by recursively choosing either the current pixel values or a percentage of the previous pixel values, summed with the current pixel values as the output pixel value. The theory is that any large pixel changes between successive frames indicate motion, and as such must be preserved in the output frame. Smaller changes are more likely caused by noise in the current frame, and therefore the previous frame can be used. This recursive action effectively reduces noise while preserving the output image content by masking small changes but preserving larger pixel changes.

The Motion Adaptive Noise Reduction LogiCORE IP is very flexible and can be used in a number of modes and configurations. While it can be used as a stand-alone core, a comprehensive set of registers and interrupts along with the provided device driver make the Motion Adaptive Noise Reduction module highly programmable and easy to control in real-time with a processor such as MicroBlaze™ processor.

Standards Compliance

The MANR core is compliant with the AXI4-Lite interconnect standard as defined in the UG761, *AXI Reference Guide*.

Feature Summary

The MANR core supports resolutions of up to 1920x1080 using 8-bit YC4:2:0 or YC4:2:2 chroma formats. This core uses a recursive temporal filter arrangement whereby the new input image is filtered with the frame that was processed by the MANR core during the previous frame period. Due to this method, temporal differences are established on a pixel-by-pixel basis. Grading the differences as noise or motion allows the core to generate an optimal output where temporary noise is reduced, but motion is conserved.

The grading of the differences is subjective and has been made programmable in the MANR core. Grading is embodied in the user-programmable motion transfer function (MTF). Typically, it is a function that maps the smaller pixel-differences as “more likely to be noise,” and so, the output pixel is a sum of the a larger proportion of the previous frame and less of the current frame. Conversely, larger pixel-differences are mapped as “more likely to be motion,” and so, the output pixel uses a larger proportion of the current frame and less of the previous frame. The proportions can be varied according to subjectivity. The

core GUI provides five preset MANR strengths as preset MTFs. Also, the software driver provided with the pCore version of this core includes the five MTF strengths and allows the user to load them using the software.

Video must be passed into the MANR using two AXI4-Stream interfaces in parallel: one for the current frame and one for the previous frame. Typically, the current frame will be provided from any live source that supports AXI4-Stream output. It may also be an AXI VDMA. The previous frame must originate from a frame buffer, and so it is best connected to an AXI-VDMA. The AXI4-Stream interface includes standard back-pressure signalling. This interconnect format should be used for connecting to other IP blocks that support AXI4-Stream.

Dynamic user-control is required when changing the noise reduction strength. When generating the MANR core, the user has the option of selecting the type of processor interface that is instantiated on the core. The first option is an EDK pCore interface that can be easily incorporated into an EDK project. Dynamic control of MTF is possible using AXI4-Lite. The second option is a General Purpose Processor interface. This option exposes the core's registers to the user. The user can wrap the exposed registers in an interface that is compliant with the systems processor.

Licensing

The Xilinx Motion Adaptive Noise Reduction 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 either the provided example design or alongside 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 functional simulation of the Motion Adaptive Noise Reduction 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

Obtaining Your License Key

This section contains information about obtaining a simulation, full system hardware, and full license keys.

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.

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license:

1. Navigate to the product page for this core.
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

Full License

To obtain a Full license key, you must purchase a license for the core. After doing so, click the "Access Core" link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Performance

For the MANR to process a complete 720p60 or 1080p30 frame within one frame period, the internal clock must be run at least at the pixel rate, or 74.25 MHz. For the MANR to process a complete 1080p60 frame within one frame period, the internal clock must be run at least at the pixel rate, or 148.5 MHz.

Maximum Frequency

The following are typical clock frequencies for the target devices.

- Virtex®-7 (-1) : 225 MHz
- Kintex™-7 (-1) : 225 MHz
- Virtex-6 (-1) : 225 MHz
- Spartan®-6 (-2) : 150 MHz

It should be noted that the above figures are typical, and have been used as target clock frequencies for the MANR in the slowest speed-grade for each device family. The figures apply to the main operation clock signal `clk`.

The maximum achievable clock frequency can vary depending on configuration.

Latency

The latency through the MANR core is fixed at 26 clock cycles. This measures the number of cycles between a value being clocked into the core and its equivalent data being delivered on the core output.

This latency does not take back-pressure exerted on the MANR core into account.

Throughput

The MANR core produces as much data as it consumes. If timing constraints are met, the throughput is equal to the rate at which video data is written into the core. In numeric terms, 1080P/60 YC4:2:2 represents an average data rate of 124.4 Mpixels/sec, or a burst data rate of 148.5 Mpixels/sec.

Resource Utilization

To help guide the user making system-level and board-level decisions, Table 1-1 through Table 1-4 show the resource usage observed for the MANR with all supported devices. This post-PAR characterization data has been collated through automated implementation of each configuration. This data will vary between implementations, and is intended primarily as a guideline.

The data in these tables applies to the GPP interface option only. When using the pCore, add approximately 800 to the LUT number and 1000 to the F/F number for all families.

Table 1-1: **Resources and Performance for Virtex-7 Devices**

Max Image Size (Pixels x Lines)	LUTs	FFs	BRAM36/18	DSP48E1	F_{MAX} (MHz)/ Speed Grade		
					-1	-2	-3
640x360	1038	882	0/1	3	242	-	-
1920x1080	1038	882	0/1	3	242	293	323

Table 1-2: **Resources and Performance for Kintex-7 Devices**

Max Image Size (Pixels x Lines)	LUTs	FFs	BRAM36/18	DSP48E1	F_{MAX} (MHz)/ Speed Grade		
					-1	-2	-3
640x360	1038	882	0/1	3	239	-	-
1920x1080	1038	882	0/1	3	239	288	337

Table 1-3: **Resources and Performance for Virtex-6 Devices**

Max Image Size (Pixels x Lines)	LUTs	FFs	BRAM36/18	DSP48E1	F_{MAX} (MHz)/ Speed Grade		
					-1	-2	-3
640x360	927	882	0/1	3	216	-	-
1920x1080	927	882	0/1	3	216	262	293

Table 1-4: **Resources and Performance for Spartan-6 Devices**

Max Image Size (Pixels x Lines)	LUTs	FFs	RAM16/8	DSP48E1	F_{MAX} (MHz)/ Speed Grade		
					-1	-2	-3
640x360	919	882	0/1	3	169	-	-
1920x1080	919	882	0/1	3	169	195	195

XILINX®

Chapter 2

Core Interfaces and Register Space

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

Port Descriptions

This section contains details about the core interfaces and signals.

Core Interfaces

The MANR core includes control interfaces and data interfaces, as described in this section. These interfaces and signals are shown in Figure 2-1.

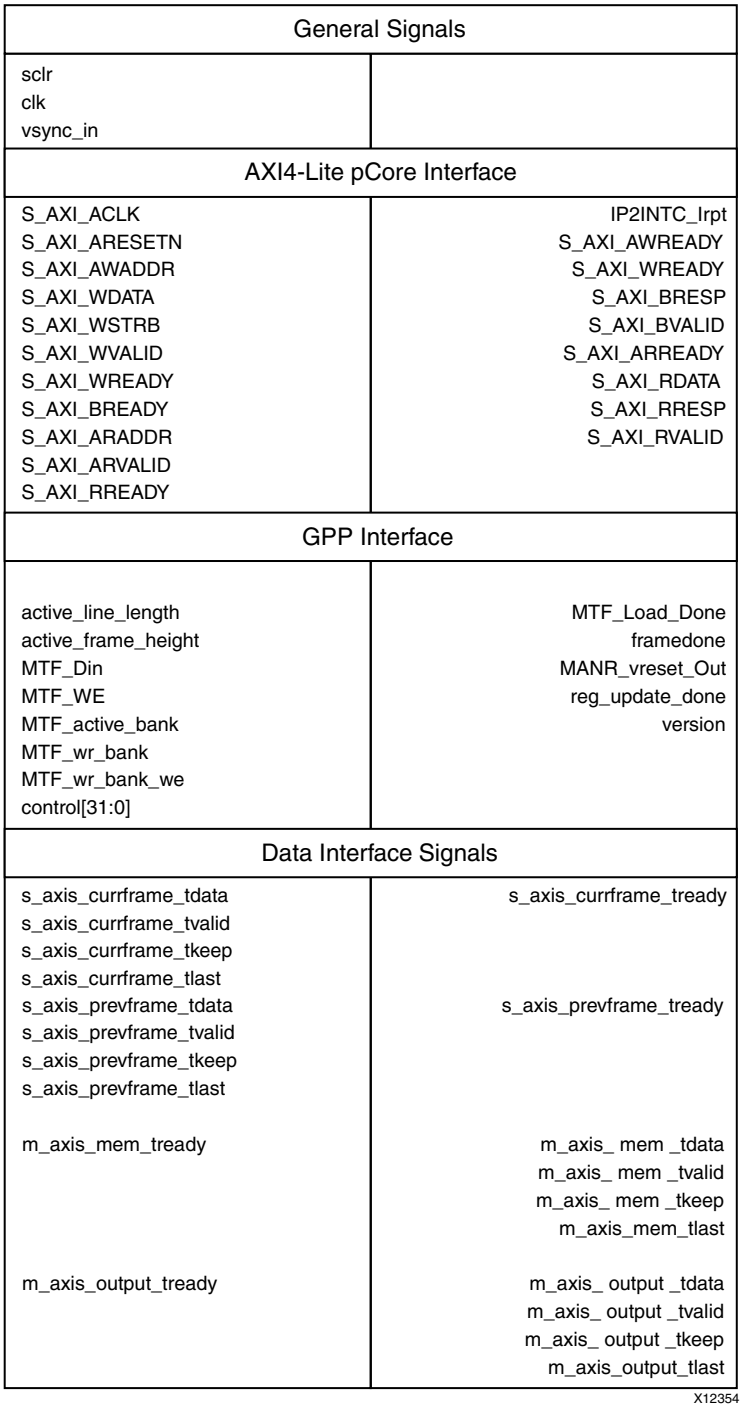

Figure 2-1: **MANR Interfaces and Signals**

Control Interfaces

Processor interfaces provide the system designer with the ability to dynamically control the parameters within the core. The MANR core supports the following two processor interface options.

- AXI4-Lite pCore Interface: The designer may select AXI4-Lite option on the MANR core if it is to be used in an EDK-based embedded system. The AXI4-Lite pCore interface creates a hardware peripheral that can be easily added to an AXI4-based EDK Project. When the core is connected to the system's AXI4-Lite interconnect, the system processor can easily access the core's registers and control the operation of the core.
- General Purpose Processor (GPP) Interface: The General Purpose Processor interface option should be selected by the user when the core is to be used in a system that does not include an AXI4 compliant system processor. The General Purpose Processor interface exposes all of the core's control and status signals. This allows the user to wrap these signals with a user-defined bus interface targeting any arbitrary processor.

Data Interface

The MANR supports the use of AXI4-Stream interfacing for I/O of all data.

Core Signal Names and Descriptions

General Interface Signals

Regardless of the type of processor interface used by the core, the MANR uses the signaling shown in Table 2-1.

Table 2-1: **General Interface Signals**

Name	Direction	Description
sclr	In	System synchronous reset (active high)
clk	In	Main core clock
vsync_in	In	Vertical sync input

Control Interface Signals

Processor interfaces provide the ability to dynamically control core parameters. The MANR core supports two processor interface options:

- AXI4-Lite pCore Interface: See Table 2-2, page 12 for the signals and descriptions.
- General Purpose Processor Interface: See Table 2-3, page 15 for the signals and descriptions.

Table 2-2: **AXI4-Lite Control Bus Signals**

Name	Direction	Description
S_AXI_ACLK	In	AXI Clock
S_AXI_ARESETN	In	AXI Reset, active low
IP2INTC_Irpt	Out	Interrupt request output
S_AXI_AWADDR	In	AXI4-Lite Write Address Bus. The write address bus gives the address of the write transaction.

Table 2-2: **AXI4-Lite Control Bus Signals *(Cont'd)***

Name	Direction	Description
S_AXI_AWVALID	In	AXI4-Lite Write Address Channel Write Address Valid. This signal indicates that valid write address is available. 1 = Write address is valid. 0 = Write address is not valid.
S_AXI_AWREADY	Out	AXI4-Lite Write Address Channel Write Address Ready. Indicates core is ready to accept the write address. 1 = Ready to accept address. 0 = Not ready to accept address.
S_AXI_WDATA	In	AXI4-Lite Write Data Bus
S_AXI_WSTRB	In	AXI4-Lite Write Strobes. This signal indicates which byte lanes to update in memory.
S_AXI_WVALID	In	AXI4-Lite Write Data Channel Write Data Valid. This signal indicates that valid write data and strobes are available. 1 = Write data/strobes are valid. 0 = Write data/strobes are not valid.
S_AXI_WREADY	Out	AXI4-Lite Write Data Channel Write Data Ready. Indicates core is ready to accept the write data. 1 = Ready to accept data. 0 = Not ready to accept data.
S_AXI_BRESP(2)	Out	AXI4-Lite Write Response Channel. Indicates results of the write transfer. 00b = OKAY - Normal access has been successful. 01b = EXOKAY - Not supported. 10b = SLVERR - Error. 11b = DECERR - Not supported.
S_AXI_BVALID	Out	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid. 1 = Response is valid. 0 = Response is not valid.
S_AXI_BREADY	In	AXI4-Lite Write Response Channel Ready. Indicates Master is ready to receive response. 1 = Ready to receive response. 0 = Not ready to receive response.
S_AXI_ARADDR	In	AXI4-Lite Read Address Bus. The read address bus gives the address of a read transaction.
S_AXI_ARVALID	In	AXI4-Lite Read Address Channel Read Address Valid. 1 = Read address is valid. 0 = Read address is not valid.

Table 2-2: **AXI4-Lite Control Bus Signals *(Cont'd)***

Name	Direction	Description
S_AXI_ARREADY	Out	AXI4-Lite Read Address Channel Read Address Ready. Indicates core is ready to accept the read address. 1 = Ready to accept address. 0 = Not ready to accept address.
S_AXI_RDATA	Out	AXI4-Lite Read Data Bus
S_AXI_RRESP(2)	Out	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer. 00b = OKAY - Normal access has been successful. 01b = EXOKAY - Not supported. 10b = SLVERR - Error. 11b = DECERR - Not supported.
S_AXI_RVALID	Out	AXI4-Lite Read Data Channel Read Data Valid. This signal indicates that the required read data is available and the read transfer can complete. 1 = Read data is valid. 0 = Read data is not valid.
S_AXI_RREADY	In	AXI4-Lite Read Data Channel Read Data Ready. Indicates master is ready to accept the read data. 1 = Ready to accept data. 0 = Not ready to accept data.

Table 2-3: **GPP Register Signals**

Name	Direction	Description
active_line_length[11:0]	In	Active line length register bits
active_frame_height[11:0]	In	Active frame height register bits
MTF_Din[7:0]	In	MTF data input
MTF_WE	In	MTF data write enable
MTF_active_bank	In	MTF active bank select 0 = Bank 1 1 = Bank 2
MTF_wr_bank	In	MTF write bank select 0 = Bank 1 1 = Bank 2
MTF_wr_bank_we	In	MTF write bank write enable
MTF_Load_Done	Out	MTF loading done: asserted when MTF has loaded
control[31:0]	In	Control register bits
framedone	Out	Indicates the end of the frame

Table 2-3: **GPP Register Signals**

Name	Direction	Description
MANR_vreset_Out	Out	General purpose reset signal asserted for one line period during vertical blanking
reg_update_done	Out	Issued during Vertical blanking when the register values have been transferred to the active registers
version	Out	Core HW version number

Data Interface Signals

The MANR core accepts video data via its AXI4-Stream interfaces. The output is always delivered through two further AXI4-Stream interfaces, as shown in Table 2-4.

Table 2-4: **AXI4-Stream Interface Signals**

Name	Direction	Description
s_axis_currframe (Video Data Source)		
s_axis_currframe_tdata	In	AXI4-Stream Video Data Input for current frame source. bits[7:0]: Luma; bits[15:8]: Chroma.
s_axis_currframe_tvalid	In	AXI4-Stream tValid input signal for current frame source. Indicates valid data on the s_axis_currframe_tdata bus.
s_axis_currframe_tlast	In	AXI4-Stream tLast input signal for current frame source. Coincident with the final pixel in a line on s_axis_currframe_tdata.
s_axis_currframe_tready	Out	AXI4-Stream tReady output signal for current frame source. High value indicates core is ready to receive data.
s_axis_currframe_tkeep	In	AXI4-Stream tKeep signal for current frame source. Input should be driven to all '1's.
s_axis_prevframe (Temporal Feedback Input from Frame Buffer)		
s_axis_prevframe_tdata	In	AXI4-Stream Video Data Input for previous frame source. bits[7:0]: Luma; bits[15:8]: Chroma.
s_axis_prevframe_tvalid	In	AXI4-Stream tValid input signal for previous frame source. Indicates valid data on the s_axis_prevframe_tdata bus.
s_axis_prevframe_tlast	In	AXI4-Stream tLast input signal for previous frame source. Coincident with the final pixel in a line on s_axis_prevframe_tdata.
s_axis_prevframe_tready	Out	AXI4-Stream tReady output signal for previous frame source. High value indicates core is ready to receive data.
s_axis_prevframe_tkeep	In	AXI4-Stream tKeep signal for previous frame source. Input should be driven to all '1's.
m_axis_mem (Temporal Feedback Output to Frame Buffer)		

Table 2-4: **AXI4-Stream Interface Signals *(Cont'd)***

Name	Direction	Description
m_axis_mem_tdata	Out	AXI4-Stream Video Data output for temporal feedback. bits[7:0]: Luma; bits[15:8]: Chroma.
m_axis_mem_tvalid	Out	AXI4-Stream tValid output signal for temporal feedback. Indicates valid data on the m_axis_mem_tdata bus.
m_axis_mem_tlast	Out	AXI4-Stream tLast output signal for temporal feedback. Coincident with the final pixel in a line on m_axis_mem_tdata.
m_axis_mem_tready	In	AXI4-Stream tReady input signal for temporal feedback. High value indicates that the downstream core is ready to receive data.
m_axis_mem_tkeep	Out	AXI4-Stream tKeep signal for temporal feedback. Output is driven to all '1's.
m_axis_output (for Optional Downstream Feed)		
m_axis_output_tdata	Out	AXI4-Stream Video Data output for downstream feed. bits[7:0]: Luma; bits[15:8]: Chroma.
m_axis_output_tvalid	Out	AXI4-Stream tValid output signal for downstream feed. Indicates valid data on the m_axis_output_tdata bus.
m_axis_output_tlast	Out	AXI4-Stream tLast output signal for downstream feed. Coincident with the final pixel in a line on m_axis_output_tdata.
m_axis_output_tready	In	AXI4-Stream tReady input signal for downstream feed. High value indicates that the downstream core is ready to receive data.
m_axis_output_tkeep	Out	AXI4-Stream tKeep signal for downstream feed. Output is driven to all '1's.

Register Space

The EDK pCore provides a memory-mapped interface for the programmable registers within the core, which are described in Table 2-5. Use of these registers is described in more detail in Chapter 4, Designing with the Core.

Table 2-5: **MANR Registers Overview**

Address (hex)	Register Name	Access Type	Description	
BASEADDR + 0x0000	MANR Control	R/W	Control	
			31:3	Reserved
			2	Enable bypass mode 1 = Disable noise reduction functionality: Output = input 0 = Enable noise reduction functionality
			1	Enable register updates 1 = MANR core will update registers on next vertical sync
			0	Enable 1 = Enable MANR core on next vertical sync
BASEADDR + 0x0004	MANR Status	R	Status	
			31:2	Reserved
			1	Register update completed 1 indicates that register values for current frame have been accepted by the core. Occurs once per frame. This bit is cleared when any value is written to the register.
			0	MTF load completed 1 indicates that 64 values have been written into the MTF LUT successfully. This bit is cleared when any value is written to the register.
BASEADDR + 0x0008	MANR Error Codes	R	Error	
			24:31	Reserved
			16:23	Reserved
			8:15	Reserved
			0:7	Reserved

Table 2-5: **MANR Registers Overview *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
BASEADDR + 0x000C	MANR Frame Status	R	Frame processing status	
			31:1	Reserved
			0	Frame done 1 indicates that an entire frame has been processed by the MANR core. This bit is cleared when any value is written to the register.
BASEADDR + 0x0014	Frame Size	R/W	Active video frame dimensions	
			31:28	Reserved
			27:16	Frame height (defaults to GUI settings)
			15:12	Reserved
			11:00	Frame width (defaults to GUI settings)
BASEADDR + 0x0018	MTF Data In	R/W	MTF load data register	
			31:8	Reserved
			0:7	MTF data
BASEADDR + 0x001C	MTF Write Bank	R/W	MTF Write bank control	
			31:1	Reserved
			0	MTF bank to be updated via MTF_data In Default 0
BASEADDR + 0x00F0	HW Version Register	R		
BASEADDR + 0x0100	Software Reset	R/W	MANR SW reset of core	
			31:1	Reserved
			0	1 resets MANR core
BASEADDR + 0x021C	GIER	R/W	Global interrupt enable register	
			31	Mask to enable global interrupts
			30:0	Reserved
BASEADDR + 0x0220	ISR	R	Interrupt status register	
			31:2	Reserved
			1	Edge sensitive interrupt for IP core done with video frame
			0	Reserved
BASEADDR + 0x0228	IER	R/W	Interrupt enable register	
			31:2	Reserved

Table 2-5: **MANR Registers Overview *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
			1	Mask or enable interrupt for IP core done with video frame
			0	Reserved

Chapter 3

Customizing and Generating the Core

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

Graphical User Interface

The Xilinx Motion Adaptive Noise Reduction (MANR) LogiCORE IP is easily configured to meet the developer's specific needs through the CORE Generator Graphical User Interface (GUI). This section provides a quick reference to parameters that can be configured at generation time. Figure 3-1 shows the GUI main screen.

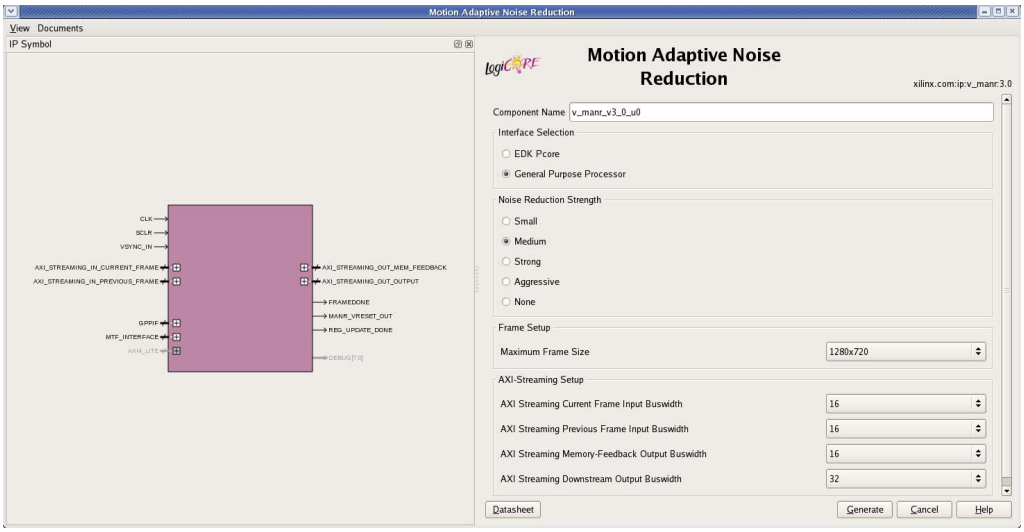

Figure 3-1: **MANR Main Screen**

The main screen displays a representation of the IP symbol on the left side, and the parameter assignments on the right side, which are described as follows:

- **Component Name:** The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, A to Z, 0 to 9 and "_".

  Note: The name "v_manr_v3_0" is not allowed.

- Interface Selection: The MANR is generated with one of two interfaces.
  - **EDK pCore Interface:** CORE Generator software generates the MANR as a pCore that can be easily imported into an EDK project as a hardware peripheral. The core registers can then be programmed in real-time via a MicroBlaze processor and the AXI4-Lite interface. When the EDK pCore is selected, the rest of the options are disabled and set to the default value. All modifications to the MANR pCore are made with the EDK GUI.
  - **General Purpose Processor Interface:** CORE Generator software generates a set of ports that can be used to program the MANR.
- **AXI Stream Input/Output Bus Widths**: There are four AXI4-Stream interfaces on this core: `s_axis_currframe`, `s_axis_prevframe`, `m_axis_mem` and `m_axis_output`. The data bus parts of these interfaces can be 16, 32, 64 or 128 bits wide, with the exception of the `m_axis_output` bus, which must be at least 32 bits wide because it includes motion data. Input and output bus widths can be different, if necessary. Care should be taken at the system level to ensure that the AXI4-Stream bus width is the same at both ends of the buses.
- **Noise Reduction Strength:** This parameter selects the default MTF. The MTF is initialized according to one of the following settings. The MTF is fully programmable, and the initial values specified during core generation can easily be overridden by programming the desired MTF at run time.
  - Small: Specifies ¼ gain exponential curve for MTF
  - Medium: Specifies ½ gain exponential curve for MTF
  - Strong: Specifies ¾ gain exponential curve for MTF
  - Aggressive: Specifies full gain exponential curve for MTF
  - None: Specifies all zeroes for MTF; bypasses noise reduction such that the output pixel always equals the input pixel.
- **Frame Setup:** This parameter configures the MANR maximum frame dimensions. The following choices are available:
  - 1920x1080
  - 1280x720
  - 720x480
  - 640x480
  - 640x360

Parameter Values in the XCO File

Table 3-1 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 3-1: **XCO Parameter Values**

XCO Parameter	Default	Valid Values
component_name	v_manr_v3_0_u0	Not v_manr_v3_0
interface_selection	EDK_Pcore	EDK_Pcore General_Purpose_Processor

Table 3-1: **XCO Parameter Values *(Cont'd)***

XCO Parameter	Default	Valid Values
m_axis_mem_tdata_width	16	16 32 64 128
m_axis_output_tdata_width	32	32 64 128
max_frame_size	1280x720	640x360 640x480 720x480 1280x720 1920x1080
noise_reduction	2	0 1 2 3 4
s_axis_currframe_tdata_width	16	16 32 64 128
s_axis_prevframe_tdata_width	16	16 32 64 128

Output Generation

The output files generated from Xilinx CORE Generator for the MANR core depend upon whether the interface selection is set to EDK pCore or General Purpose Processor. The output files are placed in the project directory. Table 3-2 shows the general files that the CORE Generator tool delivers regardless of the interface selection.

Table 3-2: **General CORE Generator Output 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.

EDK pCore Files

When the pCore is generated, the CORE Generator tool creates the following directory structure within the `<project_directory>/<component_name>` folder:

- drivers
  - manr_v1_01_a
    - data
    - doc
      - html
        - api
    - example
    - src
- pcores
  - axi_manr_v3_00_a
    - data
    - hdl
      - vhdl

<project directory>

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

<project directory>/<component_name>/pcores/axi_manr_v3_00_a/data

This directory contains files that EDK uses to define the interface to the pCore.

< project directory>/<component_name>/pcores/ axi_manr_v3_00_a /hdl/vhdl

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

< project directory>/<component_name>/drivers/manr_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>/drivers/ manr_v1_01_a /doc/html/api

This directory contains HTML documentation files for the pCore's software driver.

< project directory>/<component_name>/drivers/ manr_v1_01_a /src

This directory contains the source code of the pCore's software driver. The delivered files are listed in Table 3-3.

Table 3-3: **Software Driver Files Provided with the MANR pCore**

File Name	Description
xmanr.c	Provides the API access to all of the features of the Xilinx MANR device driver.
xmanr.h	Provides the API access to all of the features of the Xilinx MANR device driver.

Table 3-3: **Software Driver Files Provided with the MANR pCore**

File Name	Description
xmanr_g.c	Contains a template for configuration table of Xilinx MANR devices.
xmanr_hw.h	Contains identifiers and register-level driver functions (or macros) that can be used to access the Xilinx MANR device.
xmanr_intr.c	Contains interrupt-related functions of Xilinx MANR device driver.
xmanr_sint.c	Contains static initialization methods for Xilinx MANR device driver.
example.c	Examples that demonstrate how to control the Xilinx MANR.

General Purpose Processor Files

When the interface selection is set to General Purpose Processor, the CORE Generator tool 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 of the files listed in Table 3-4.

Table 3-4: **CORE Generator Files for GPP Mode**

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 templates for instantiating the core.
<component_name>.v <component_name>.vhd	The structural simulation models for the core. They are used for functionally simulating the core.
<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.tx	A text file listing all of the output files produced when the customized core was generated in the CORE
<component_name>.asy	IP symbol file.
<component_name>.gise <component_name>.xise	ISE software subproject files for use when including the core in ISE software designs.

XILINX®

Chapter 4

Designing with the Core

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

Theory of Operation

The noise reduction algorithm is implemented with a recursive temporal filter that uses a programmable motion transfer function (MTF) to control both the shape of the noise reduction curve, as well as the "strength" of the noise reduction. The theory of this filter is simple and consists of two operations.

First, the motion value for the current pixel is calculated by taking the absolute value of the difference between the current and previous pixels. This value is then filtered through a fixed coefficient FIR filter to form a scalar value representing the motion value present in the pixel for the current video frame. This motion value is used as an index to the MTF look-up table.

Second, the value generated from the MTF is used as a multiplier to scale the difference value. The resulting value is summed with the current frame pixel value, resulting in an output pixel that contains a percentage of the previous frame and the current frame. This same output is then written to memory and becomes the previous frame for the next cycle, thus forming a recursive filter. Consequently, the entire input frame is filtered in a recursive fashion as shown in Figure 4-1.

For the MANR v3.0 core, the above operation is carried out independently for luma and chroma channels. A separate engine is included for this purpose. The sub-sampled Cr and Cb channels use this second engine. Switching between Cr and Cb is handled internally.

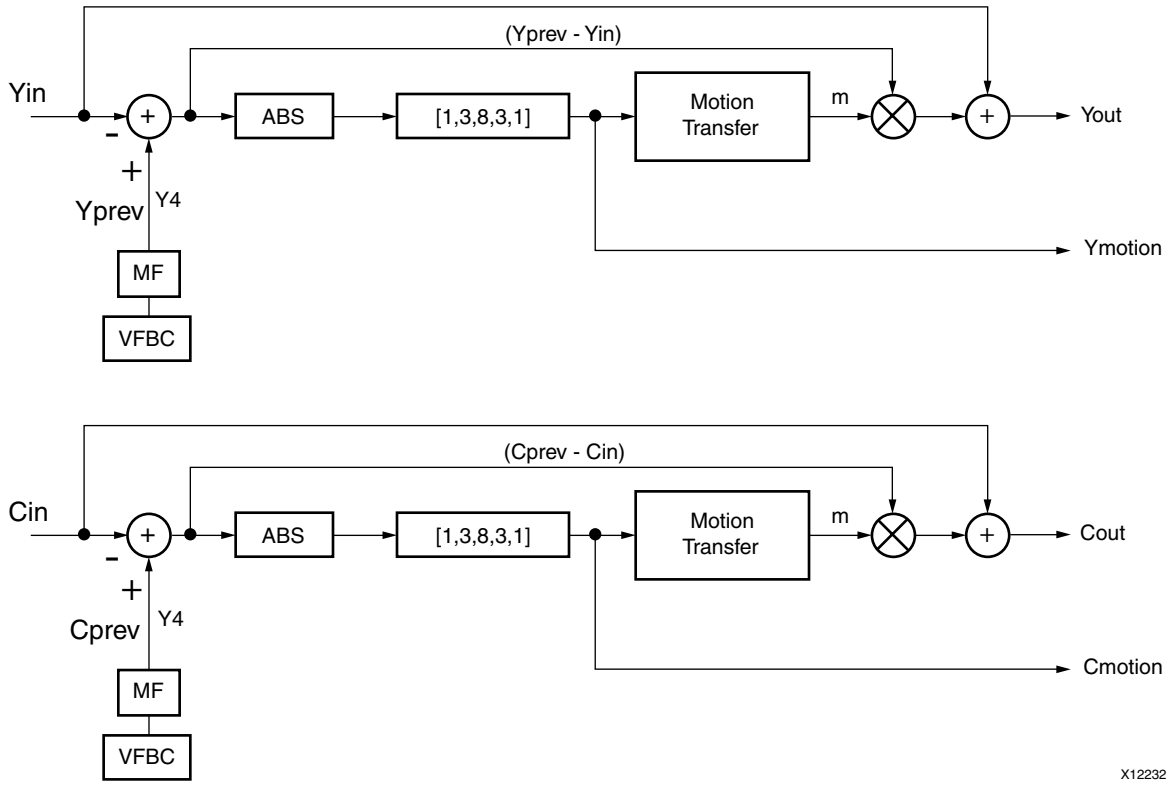

Figure 4-1: **Motion Adaptive Noise Reduction**

The key to successful noise reduction lies in the choice of the MTF. Any monotonically decreasing function can be loaded into the MTF. However, certain functions lend themselves well to this application and have been used in the industry. Two such functions are the exponential and Gaussian. The MANR LogiCORE IP is initialized from CORE Generator using an "exponential" shape for the MTF. This shape is then attenuated to provide the different possible noise reduction strengths available. The exponential shape provided has been shown to be effective at reducing noise while minimizing "smearing" or "ghosting" caused by the recursive nature of the filter.

The exponential transfer function is shown in Figure 4-2. The Y-axis denotes the amount of recursion, and the X-axis denotes the amount of motion.

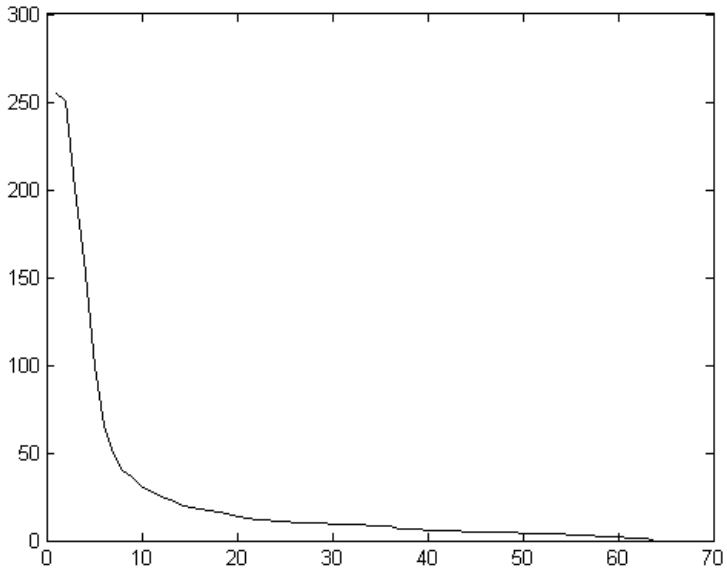

Figure 4-2: **Exponential MTF**

The function shown is monotonically decreasing. This implies that the amount of recursion is inversely proportional to the amount of motion detected. For example, a large motion value of 63 would result in an output of 0 from the MTF. This would result in none of the previous pixel data being applied to the output data. Conceptually, this makes sense. A large motion value indicates that the pixel changes are most likely not due to noise; therefore the output image should consist of mostly or all of the current input image. Conversely, a small motion value results in a large output value from the MTF, and hence more recursion. Logically this follows since small changes in the pixels from frame to frame are more likely due to noise than motion, and hence more of the previous image should be used to form the output image. The function also has a "knee" or "shelf" at the beginning of the curve. This maximizes recursion in the area of the curve where noise is most likely to occur, but still rolls off quickly as the magnitude of the luma changes increase (indicating that actual motion is present).

Using this same shape, several "strengths" of noise reduction can be realized by applying an attenuation factor to the curve in Figure 4-2. This results in the same shape response, but varying degrees of recursion for the same shape. Shown in Figure 4-3 are the exponential MTFs with an attenuation of 0.75, 0.5, 0.25 and zero applied. The zero case is also called "none" or "bypass" because regardless of the motion scalar value per pixel, the output of the filter will always be the current pixel, that is, the motion transfer function of zero results in no recursion. To ease selection of a noise reduction strength setting, each curve has been assigned a qualitative value of aggressive, strong, medium, weak, and none. Each curve is shown in Figure 4-3. These settings map directly to the selections available in the Core Generator GUI. Selecting a particular strength initializes the MTF on power-up with that setting. The power-up MTF can always be overwritten at run-time.

In Figure 4-3, the curves are shown relative to the aggressive setting to illustrate how the attenuation factor is applied. For reference:

- The aggressive curve is shown as a solid line
- The strong setting is shown as a dotted line
- The medium setting is shown as a dashed line
- The weak setting is shown as a "dash-dot" line

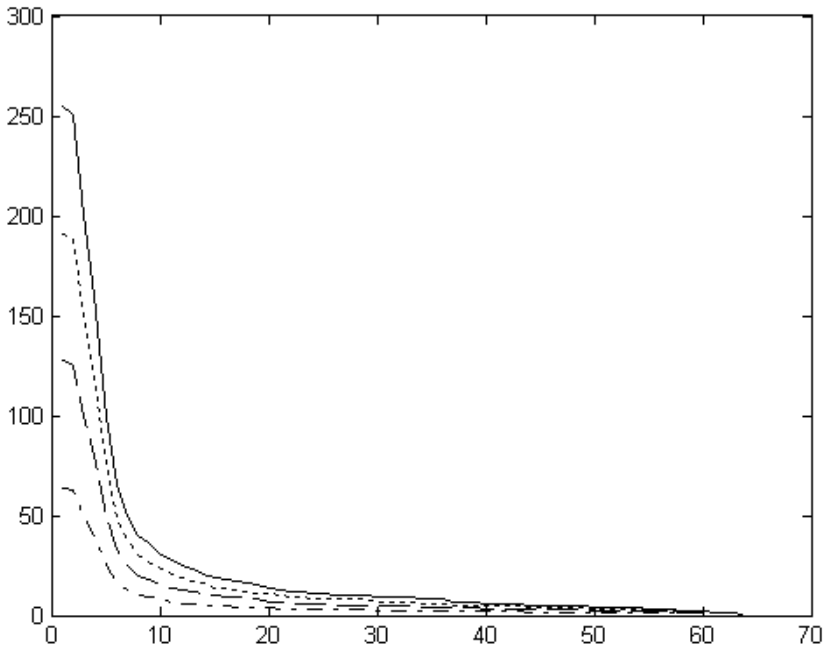

Figure 4-3: **MTF Settings**

The MANR core supports two MTF tables in memory. Only one table can be active in a given frame period. See MTF Storage and Switching in Chapter 4 for details.

General Design Guidelines

MTF Storage and Switching

The MTF values are stored in RAM internal to the MANR core. Two separate banks of memory are supported enabling two separate MTF curves. Storing two different MTFs may be useful in situations where the content being filtered differs in motion content. For example, a source may switch between a camera showing a fixed scene with little movement, to a more complex scene with many moving objects. One MTF can be optimized for noise reduction, while the other can balance noise reduction and motion artifact from recursion. For example, the "Aggressive" exponential curve shown in Figure 4-3 could be made active when the scene has little motion (since this curve will have more recursion, and hence more smearing artifacts) while the "Medium" curve could be used when the material has a large motion content. When the source is switched, the MTF bank can also be switched, allowing a quick optimization of the MTF for a given source.

In addition, the MTF values can be updated on a frame-by-frame basis, allowing a microprocessor to easily control and optimize the MTF based on the expected source material and other conditions.

A key advantage of the Xilinx MANR core is the ability to define and use your own MTF curves. To do this successfully, a few properties of MTFs should be well understood.

Each MTF consists of 64 discrete values that form a piecewise linear definition of the MTF curve. For example, using the "aggressive" exponential curve included with the core, the MTF data would appear as in the following table, where "Address" is the offset into the MTF memory from the base address and "Value" is the 8-bit value.

The addressing is very important. Recall that the MTF must be monotonically ***decreasing***. This means that for large motion values, the MTF should output a small value; for small motion values, the MTF should output a large value. In addition, for the register bypass mode to work, MTF value at address 63 must be zero.

Address	Value	Address	Value	Address	Value	Address	Value
0	255	8	36	16	17	24	11
1	250	9	30	17	16	25	10
2	200	10	28	18	15	26	10
3	160	11	25	19	14	27	10
4	100	12	23	20	13	28	10
5	65	13	21	21	12	29	9
6	50	14	19	22	12	30	9
8	40	15	18	23	11	31	9

Address	Value	Address	Value	Address	Value	Address	Value
32	9	40	6	48	5	56	3
33	8	41	6	49	4	57	2
34	8	42	6	50	4	58	2
35	8	43	5	51	4	59	2
36	7	44	5	52	4	60	1
37	7	45	5	53	4	61	1
38	7	46	5	54	3	62	1
39	6	47	5	55	3	63	0

Input Interfaces

All video data is passed into the MANR v3.0 core through two AXI4-Stream interfaces. The intended use of the MANR core is to simultaneously access two frames that differ temporally by one frame period. These frames are referred to as the "current" and "previous" frames.

The current frame is accessed through the S_AXIS_CURRFRAME AXI4-Stream interface. The previous frame is accessed through the S_AXIS_PREVFRAME AXI4-Stream interface. Typically, the data source for this interface is a frame buffer. The MANR output frame must be fed back via external frame-buffer memory in order to become the previous frame during the next frame period.

Both of these interfaces handle 8-bit YC data, transmitted as the lowest 16 bits of the tData element of the input AXI4-Stream bus. Luma occupies bits 7:0; and chroma occupies bits 15:8.

The MANR uses internal FIFOs and the AXI4-Stream flow-control in order to synchronize incoming data from these two interfaces.

Output Interfaces

Video data is passed out from the MANR v3.0 core through two AXI4-Stream interfaces. Since the MANR operates as a recursive temporal filter, the output frame must be written into memory, where it must become available as the previous frame during the next frame period. The M_AXIS_MEM AXI4-Stream interface should be used for writing the frame to the frame buffer. This interface handles 8-bit YC data, transmitted as the lowest 16-bits of the tData element of the output AXI4-Stream bus. Luma occupies bits 7:0, and chroma occupies bits 15:8.

In addition to writing the data back to memory, the same processed output video data is made available on the M_AXIS_OUTPUT AXI4-Stream interface, for use by downstream processing blocks. The interface handles 8-bit YC data, transmitted as the lowest 16-bits of the tData element of the output AXI4-Stream bus. Luma occupies bits 7:0, and chroma occupies bits 15:8. In addition, the M_AXIS_OUTPUT_TDATA bus also holds the motion data that it has calculated for use by downstream processing blocks. The luma motion occupies bits 23:16 of this output bus. The chroma motion occupies bits 31:24.

Figure 4-4 shows a typical use-case including the use of the AXI-VDMA block for external memory access.

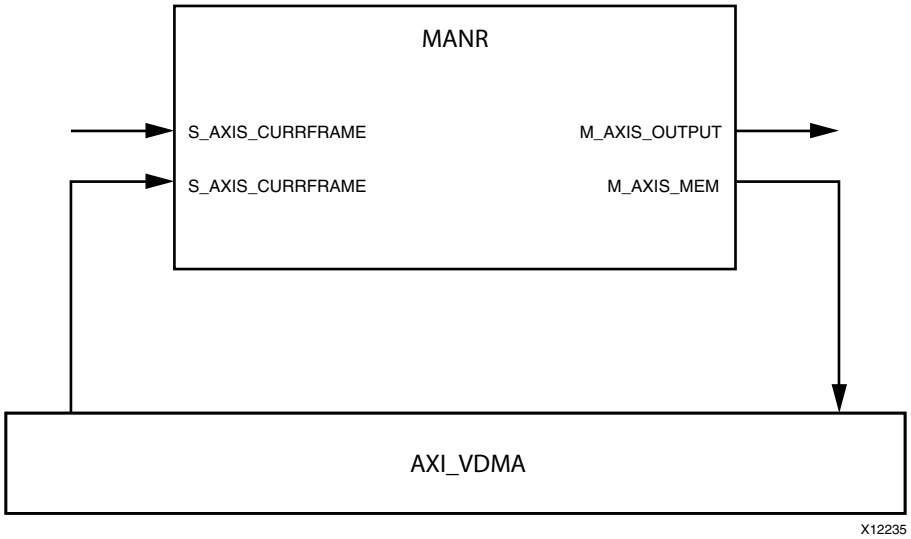

Figure 4-4: **Typical MANR Connectivity**

pCore Interrupts

The MANR pCore provides an internal interrupt controller with masking and enable to make interrupt handling easier.

The MANR generates a single interrupt "frame done" indicating that it has finished processing the current frame. This signal can be useful for software to manage the core in the context of a larger pipeline.

Use Models

The Motion Adaptive Noise Reduction LogiCORE IP is a versatile core that can be used in myriad ways. Two examples are provided that show the core usage for noise reduction only, and as the noise-reduction engine and motion-detection engine for a larger system.

It is important to note that regardless of the application, the MANR core must have access to external memory using AXI VDMA blocks. The recursive nature of the filter requires that the current output frame of the core be written to memory to be stored and used as the previous frame for the next set of calculations.

In Figure 4-5 and Figure 4-6, thick lines are used to indicate video data flow in the system.

Use Model 1: Noise Reduction Application

Figure 4-5 shows an example where the MANR is used alone to reduce noise and leave the image in memory. In this case, all video data (current and previous frames) is fed to the MANR from the memory using the AXI VDMA block. Such a system can be built using the building blocks provided by Xilinx (AXI VDMA, Timing Controller, OSD, etc.).

In Figure 4-5, video data originates from the camera shown on the left. The data is preprocessed (for example, color-space-conversion and chroma resampling to create

YC422 data) before being written into memory. Further processing can be undertaken in this domain, before writing the data into the memory using the AXI-VDMA.

The second AXI-VDMA in Figure 4-5 reads the data coming out and going to the MANR core as the current frame.

The third AXI-VDMA in Figure 4-5 handles the temporal feedback path. It takes the MANR output for storage as the previous frame input. Also, the bidirectional AXI-VDMA feeds the previous frame back into the core.

The fourth AXI-VDMA in Figure 4-5 also reads the processed frame and passes it out through the OSD via optional post-processing as desired. This shows the noise-reduced image being used for both temporal feedback and display purposes.

The timing controller shown in Figure 4-5 is responsible for distributing video synchronization signals to all cores.

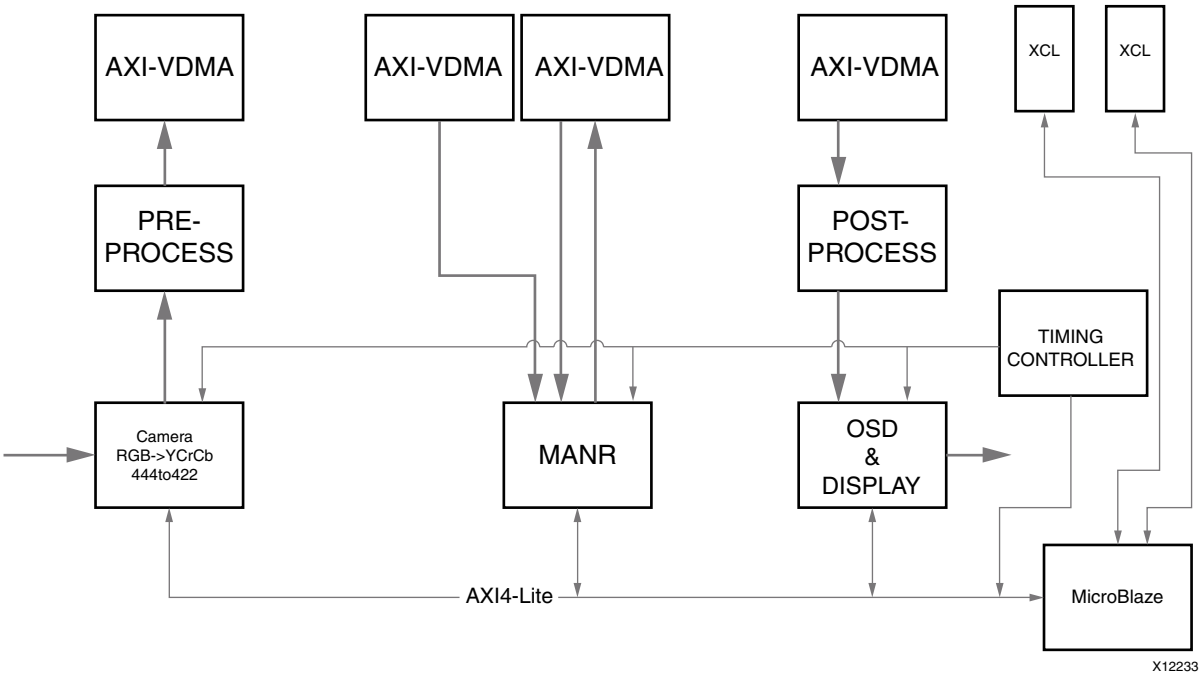

Figure 4-5: **Simple Noise Reduction**

Use Model 2: Noise Reduction and Motion Detection

In Figure 4-6, the MANR is used to calculate and provide motion data and noise reduction in a simple video processing system. The link from the MANR to the (generic) Image Processing block includes video data and pixel motion information which can be used by this target block.

In this example, as in Figure 4-5, the MANR core noise-reduces the incoming video from the camera.

However, in Figure 4-6, the MANR core also provides the video and motion data to a processing module via the AXI4-Stream output port M_AXIS_OUTPUT for additional processing of the motion and image data. Typical examples include an Image Characterization block (which makes use of the video data and the motion data outputs) or a Video Scaler block (which only uses the video data).

In this example, the image-processing block provides its output back into the external memory using the spare write-port on the second AXI-VDMA. The image passed through the OSD block, and out to a monitor may be either:

- The direct output from the MANR core (fed into the memory as shown into the third AXI-VDMA) or
- The output of the Image Processing, which is also stored in memory.

This choice can be made using application software, controlling the frame-buffer pointers that are programmed into the fourth AXI-VDMA.

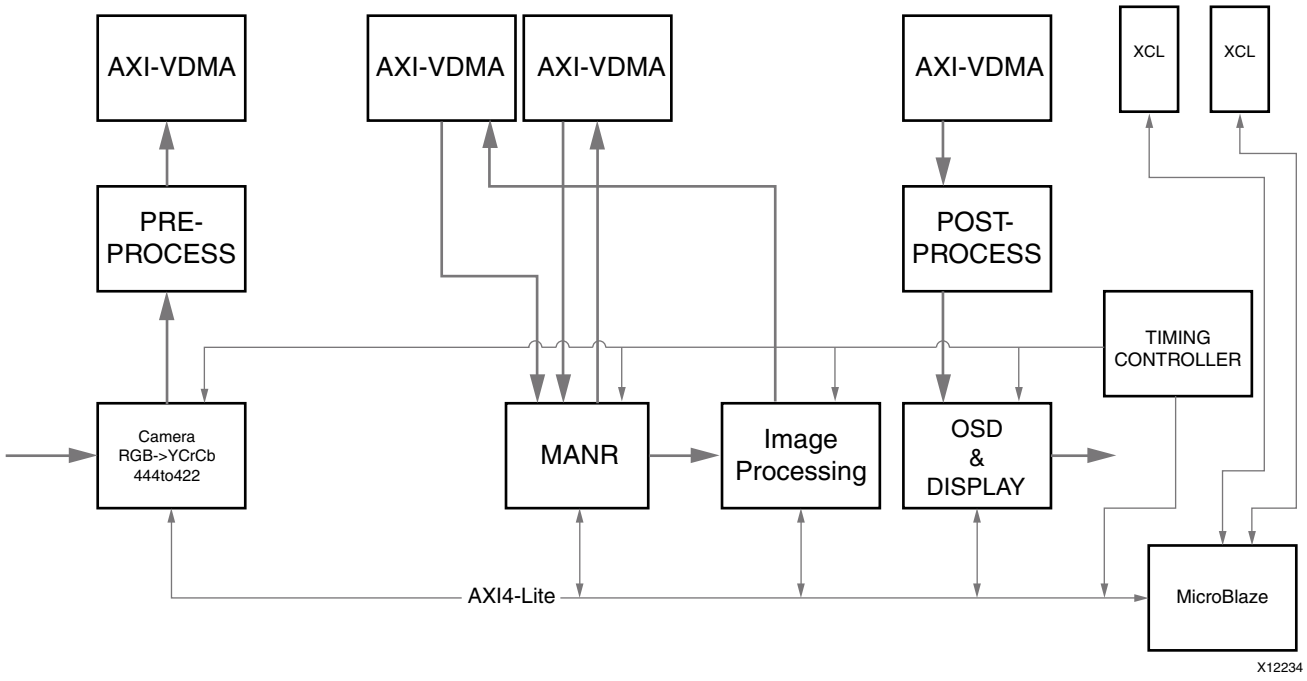

Figure 4-6: **Noise Reduction and Motion Processing**

Clocking

The MANR core has one clock (`clk`) that is used to clock the datapath of the entire core.

MANR Control and Timing

The initialization and startup of the MANR is very simple. After reset, the user need only initialize the registers to set up the frame size. Next, loading of an MTF can be done if the defaults chosen during core generation are not sufficient. Finally, the MANR is enabled and begins to process data. The following flow chart shows this process. It is important to insure that the frame buffers have been initialized prior to enabling the MANR. Otherwise, the recursive nature of the filter can result in video artifacts being propagated. For example, if prior to starting the MANR, the previous frame buffer location is loaded with invalid data, the MANR will recursively add this invalid data back into the image infinitely. To avoid this, ensure that the current and previous frame buffer locations are initialized with valid video data prior to enabling the MANR. This can be accomplished either by enabling the bypass mode in the control register, or by loading an MTF of all

zeroes for a few frames. Once a few frames have been processed, the MTF can be updated or the bypass mode disabled. See Figure 4-7.

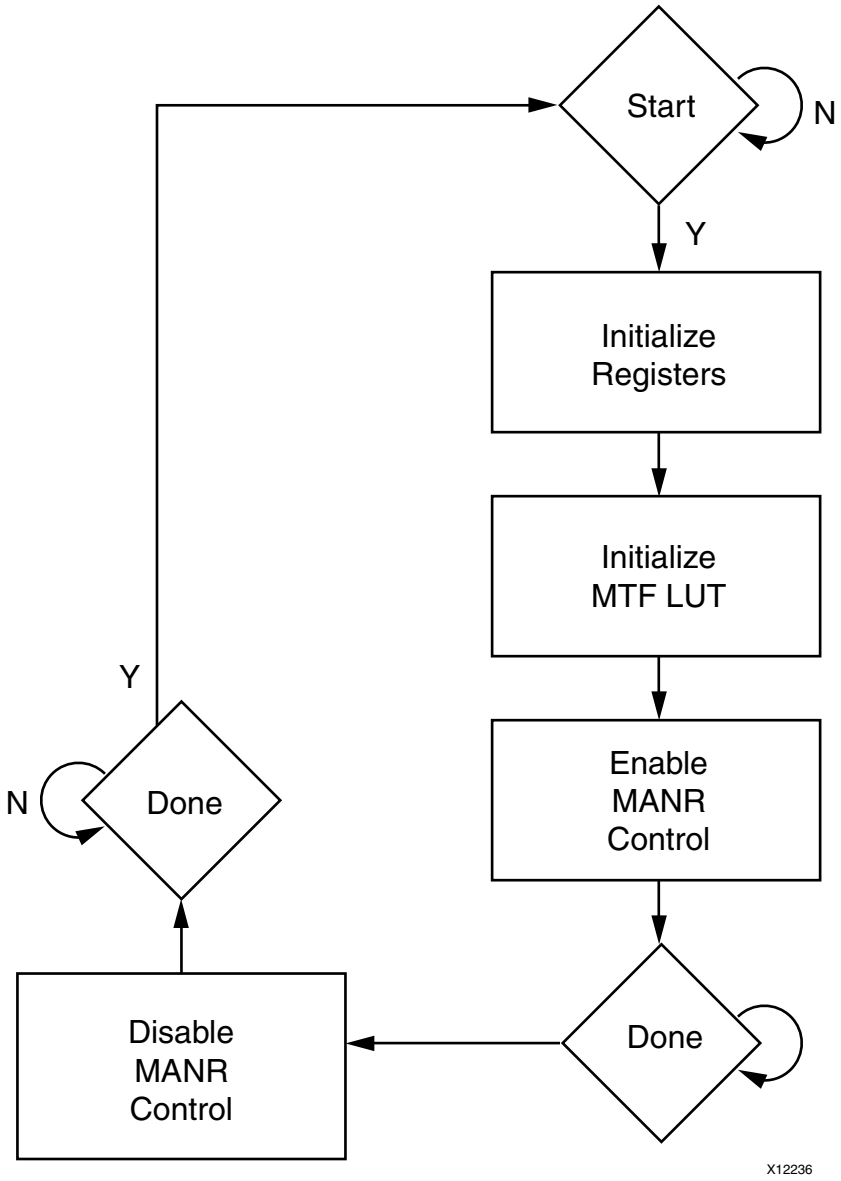

Figure 4-7: **MANR Initialization**

MTF Interface Timing

The user loads the desired motion transfer function into the Motion Transfer LUT via the `MTF_DIn` port. Loading the MTF involves writing 64 8-bit unsigned values within the range 0 through 255. The MSB is bit 7.

There are two banks available for the Motion Transfer Function. You need not have MTFs loaded into both banks. However, it is important to make sure that the correct bank is selected if you have only one bank loaded. Bank switching and selection is handled by asserting the appropriate register bits in the MANR core.

1. The active bank is indicated by driving the `MTF_Active_Bank` register accordingly. The MANR core uses this bank until `vsync` occurs after the register value is changed.
2. When updating a bank, the target bank is indicated by writing to the `MTF_Write_Bank` register. Writing any value to this register resets the internal MTF loading process.
3. The 64 values must be loaded sequentially, starting at element 0.
4. Following successful transfer of 64 MTF values, the `MTF_Load_Done` status bit is set high. If this does not occur, the load process should be re-attempted from element 0, starting with reset as directed in step 2.

The MTF loading interface is an asynchronous interface. A high level on the `MTF_WE` signal is used to capture the MTF values delivered on `MTF_Din`. An internal state-machine detects the 3rd clock cycle when `MTF_WE` is stable and high. At this point, the data is registered into the internal MTF memory. Xilinx recommends that the `MTF_WE` pulse be no less than the equivalent of six clock periods in duration. It is also required that it be low for a period no less than six clock periods in duration between write operations. Figures 4-8 and 4-9 show the timing relationships for the MTF interface. Figure 4-8 shows the detailed timing for two MTF values. Figure 4-9 shows the loading of all 64 values for the MTF.

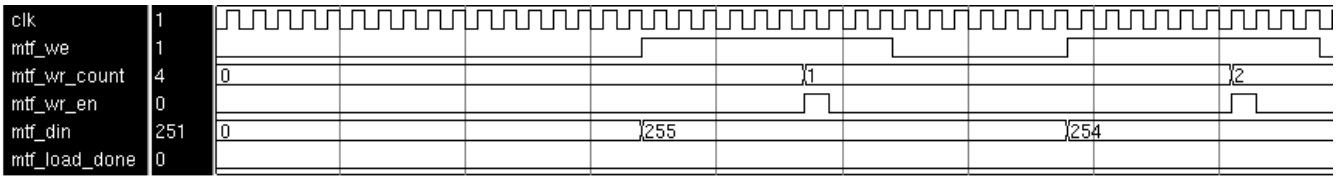

Figure 4-8: **MTF Port Timing**

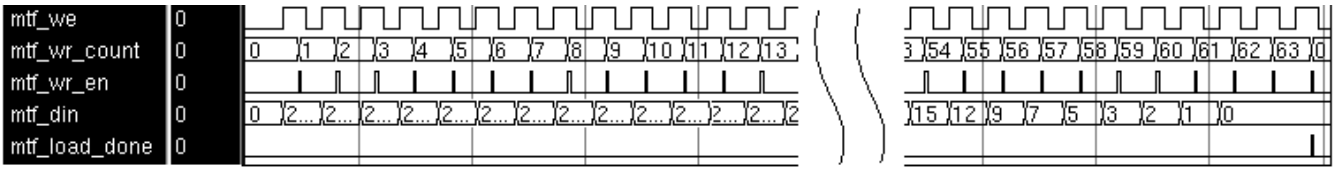

Figure 4-9: **MTF Load Timing**

Resets

The MANR core has one reset (`sclr`) that is used for the entire core. In the GPP version of the core (not EDK pCore), the signal is exposed to the user and is active High. For the pCore version, an internal software reset drives this signal (active Low).

Protocol Description

For the pCore version of the MANR core, the register interface is compliant with the AXI4-Lite interface. The video input and output interfaces are compliant with AXI4-Stream protocol.

Evaluation Core Timeout

When generated with an Evaluation Hardware license, the core includes a timeout circuit that disables the core after a specific period of time. The timeout circuit can only be reset by reloading the FPGA bitstream. The timeout period for this core is set to approximately

eight hours for a 75 MHz clock. Using a faster or slower clock changes the timeout period proportionally. For example, using a 150 MHz clock results in a timeout period of approximately four hours.

After the timeout period has expired, video output will no longer be available at the outputs of the core.

Chapter 5

Constraining the Core

This chapter contains applicable constraints for the MANR core.

Required Constraints

There are no required constraints for the MANR core.

Device, Package, and Speed Grade Selections

Device, package and speed-grade should be selected according to the worst-case throughput scenario required by the user. For more information, see Performance in Chapter 1.

Clock Frequencies

The core clock (`clk`) needs to be constrained to the frequency that the user expects to run at. Typically this is covered by system-level constraints.

Clock Management

The MANR contains no Clock Managers, DCMs, PLLs, etc. All clocks must be driven into the core from an appropriate source.

Clock Placement

There are no specific Clock placement requirements for the MANR core.

Banking

There are no specific Banking requirements for the MANR core.

Transceiver Placement

The MANR includes no transceivers.

I/O Standard and Placement

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

XILINX®

Chapter 6

Detailed Example Design

This chapter provides an example system that includes the MANR core. Important system-level aspects when designing with the MANR are highlighted, including:

- MANR usage with the Xilinx AXI-VDMA
- Inclusion of the MANR in an EDK project
- Typical usage of MANR in conjunction with other cores

Example System General Configuration

The system input and output is expected to be no larger than 720P (1280Hx720V) with a maximum pixel frequency of 74.25 MHz and equivalent clocks. The details of the example system are as follows:

- MicroBlaze processor controls MANR MTF, and image size according to user input.
- MicroBlaze processor controls scale factors according to user input.
- The system can upscale or downscale.
- When down scaling, the full input image is scaled down and placed in the center of a black 720P background and displayed
- When upscaling, the center of the 720P input image is cropped from memory and upscaled to 720P. It is then displayed as a full 720P image on the output.
- Upscaled noise is typically is very noticeable. The MANR core, placed before the scaler, removes noise when enabled. Enabling and disabling the MANR core makes a very good demonstration of it's capability.
- Operational clock frequencies are derived from the input clock.
- The user may crop a small part of the frame in memory before passing it through the MANR and scaling it. In this case, the Scaler, MANR and AXI-VDMA0 cores all need to be informed of the dimensions of the rectangle being processed. This is achieved using a MicroBlaze processor.

Figure 6-1 shows the MANR and other cores incorporated into the system described. Here are the essential details:

- The Timebase Controller is a software-configurable timing detector and generator block that generates timing signals for distribution around the system. See PG016, *LogiCORE IP Video Timing Controller Product Guide* for more information.
- The Video Scaler is a software-configurable unit that can resize video frames in real-time. See PG009, *LogiCORE IP Video Scaler Product Guide* for more information.
- The On-Screen Display (OSD) block aligns the data read from memory with the timing signals and presents it as a standard-format video data stream. It also

alpha-blends multiple layers of information (for example, text and other video data). See PG010, *LogiCORE IP On-Screen Display Product Guide* for more information.

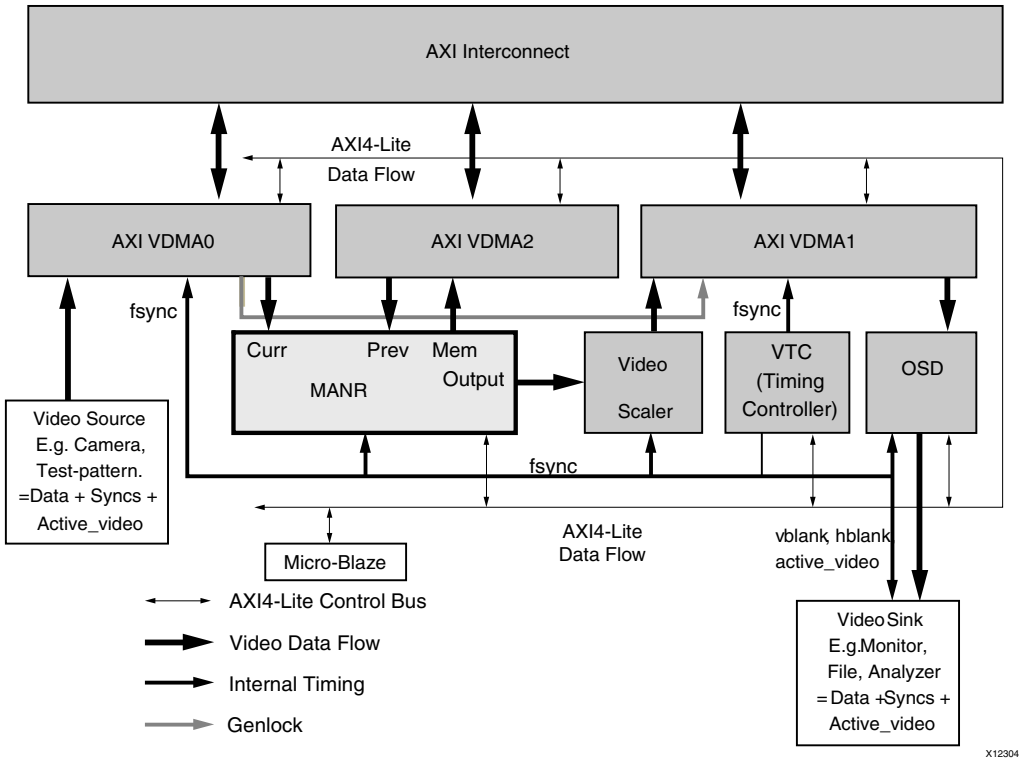

Figure 6-1: **Simplified System Diagram**

Control Buses

In this example, the MicroBlaze processor is configured to use the AXI4-Lite bus. The AXI-VDMAs, MANR, Video Scaler, Timing Controller, and OSD cores use AXI4-Lite interfaces.

AXI VDMA0 Configuration

AXI_VDMA0 is used bi-directionally. The input side takes data from the source domain and writes frames of data into DDR memory. The output side reads data (on a separate clock domain and separate video timing domain) and feeds it to the current frame input on the MANR.

The system operates using a genlock mechanism. A rotational five-frame buffer is defined in the external memory. Using the genlock bus AXI VDMA0 tells AXI VDMA1 which of the five frame locations is being written to avoid R/W collisions.

In the EDK MHS File Text, page 40, AXI VDMA0, is sourced from an engineering test-pattern generator (not included). Data is passed between the IP and AXI_VDMA0 using an AXI4-Stream interface.

AXI VDMA1 Configuration

AXI_VDMA1 is used bi-directionally. The input side takes scaled data from the Video Scaler core and writes frames of data into DDR memory. The output side reads frames from DDR memory and feeds them to the OSD core.

In the EDK MHS File Text, page 40, AXI_VDMA1 is a genlock slave to AXI_VDMA0. Data is passed between the IP and AXI_VDMA0 using an AXI4-Stream interface .

AXI VDMA2 Configuration

AXI_VDMA2 is used bi-directionally. The input side takes the MANR output from its `m_axis_mem` bus and writes frames of data into DDR memory. The output side reads frames from DDR memory and feeds them back to the MANR as the previous frame input, using `s_axis_prevframe`. The external memory uses a separate frame buffer for this feedback path.

Data is passed between the IP and AXI_VDMA2 using an AXI4-Stream interface.

MANR Configuration

The MANR core is connected so that the current frame is fed from AXI VDMA0. AXI VDMA2 is used for the temporal feedback path. Meanwhile, the MANR output is fed directly to the Video Scaler core. Note how any backpressure asserted by the scaler/ AXI-VDMA1 is asserted back on the MANR core using the AXI4-Stream path.

The MTF is loaded using MicroBlaze processor.

The MANR core uses a 148.5 MHz clock generated by the clock generator.

Video Scaler Configuration

The Video Scaler core is configured as follows:

- Single-engine 4:2:2
- 11Hx11V-taps
- 64 phases
- Shared YC coefficients
- Memory Mode
- `m_axis_tdata_width` = 16
- `s_axis_tdata_width` = 32 This is required so that the 32-bit MANR output AXI4-Stream data bus can be accepted. In this case, the scaler simply ignores the motion data bits (bits 31:16).

This core uses a 148.5 MHz clock generated by the clock generator.

EDK MHS File Text

This section contains an example EDK MHS file insert for the system described.

Note: This is NOT a complete design, but gives some idea as to the construction of a MANR system in EDK.

```
BEGIN axi_vdma
```

```
 PARAMETER INSTANCE = axi_vdma_0
 PARAMETER HW_VER = 4.00.a
 PARAMETER C_BASEADDR = 0x7e220000
 PARAMETER C_HIGHADDR = 0x7e22ffff
 PARAMETER C_USE_FSYNC = 1
 PARAMETER C_MM2S_GENLOCK_MODE = 1
 PARAMETER C_MM2S_LINEBUFFER_DEPTH = 4096
 PARAMETER C_S2MM_LINEBUFFER_DEPTH = 4096
 PARAMETER C_MM2S_MAX_BURST_LENGTH = 256
 PARAMETER C_S2MM_MAX_BURST_LENGTH = 256
 PARAMETER C_PRMRY_IS_ACLK_ASYNC = 1
 PARAMETER C_S_AXIS_S2MM_TDATA_WIDTH = 16
 PARAMETER C_M_AXIS_MM2S_TDATA_WIDTH = 16
 PARAMETER C_NUM_FSTORES = 5
 PARAMETER C_INCLUDE_SG = 0
 PARAMETER C_S2MM_LINEBUFFER_THRESH = 2560
 PARAMETER C_INCLUDE_MM2S_SF = 1
 PARAMETER C_INCLUDE_S2MM_SF = 1
 PARAMETER C_MM2S_LINEBUFFER_THRESH = 1920
 PARAMETER C_FLUSH_ON_FSYNC = 1
 BUS_INTERFACE M_AXIS_MM2S = axi_vdma_0_M_AXIS_MM2S
 BUS_INTERFACE S_AXIS_S2MM = tpg_0_M_AXIS_S2MM
 BUS_INTERFACE S_AXI_LITE = axi4lite_0
 BUS_INTERFACE M_AXI_MM2S = axi4_0
 BUS_INTERFACE M_AXI_S2MM = axi4_0
 PORT s2mm_fsync = xsvi2axi_0_fsync_out
 PORT s2mm_frame_ptr_out = axi_vdma_0_s2mm_frame_ptr_out
 PORT mm2s_frame_ptr_in = axi_vdma_0_s2mm_frame_ptr_out
 PORT mm2s_introut = axi_vdma_0_mm2s_introut
 PORT s2mm_introut = axi_vdma_0_s2mm_introut
 PORT s_axi_lite_aclk = clk_50_0000MHzMMCM0
 PORT m_axi_mm2s_aclk = clk_200_0000MHzMMCM0
 PORT m_axi_s2mm_aclk = clk_200_0000MHzMMCM0
 PORT s_axis_s2mm_aclk = vid_in_clk
 PORT m_axis_mm2s_aclk = clk_75_0000MHzMMCM0
 PORT s2mm_prmry_reset_out_n = XSVI2AXI_S2MM_RESET_OUT_N
END

BEGIN axi_vtc
 PARAMETER INSTANCE = axi_vtc_1
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_BASEADDR = 0x7ee00000
 PARAMETER C_HIGHADDR = 0x7ee0ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE XSVI_OUT = axi_vtc_1_XSVI_OUT
 PORT video_clk_in = clk_75_0000MHzMMCM0
 PORT IP2INTC_Irpt = timebase_1_IP2INTC_Irpt
 PORT fsync = axi_vtc_1_fsync
END

BEGIN axi_manr
 PARAMETER INSTANCE = axi_manr_0
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_BASEADDR = 0x7e260000
 PARAMETER C_HIGHADDR = 0x7e26ffff
 PARAMETER C_S_AXIS_CURRFRAME_TDATA_WIDTH = 16
 PARAMETER C_S_AXIS_PREVFRAME_TDATA_WIDTH = 16
 PARAMETER C_M_AXIS_MEM_TDATA_WIDTH = 16
 PARAMETER C_M_AXIS_OUTPUT_TDATA_WIDTH = 32
```

```
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE S_AXIS_CURRFRAME = axi_vdma_0_M_AXIS_MM2S
 BUS_INTERFACE S_AXIS_PREVFRAME = axi_vdma_2_M_AXIS_MM2S
 BUS_INTERFACE M_AXIS_MEM = axi_manr_0_M_AXIS_MEM
 BUS_INTERFACE M_AXIS_OUTPUT = axi_manr_0_M_AXIS_OUTPUT
 PORT S_AXI_ACLK = clk_50_0000MHzMMCM0
 PORT clk = clk_75_0000MHzMMCM0
 PORT vsync_in = axi_vtc_1_fsync
END

BEGIN axi_vdma
 PARAMETER INSTANCE = axi_vdma_2
 PARAMETER HW_VER = 4.00.a
 PARAMETER C_BASEADDR = 0x7e240000
 PARAMETER C_HIGHADDR = 0x7e24ffff
 PARAMETER C_USE_FSYNC = 1
 PARAMETER C_MM2S_GENLOCK_MODE = 1
 PARAMETER C_MM2S_LINEBUFFER_DEPTH = 4096
 PARAMETER C_S2MM_LINEBUFFER_DEPTH = 4096
 PARAMETER C_MM2S_MAX_BURST_LENGTH = 256
 PARAMETER C_S2MM_MAX_BURST_LENGTH = 256
 PARAMETER C_PRMRY_IS_ACLK_ASYNC = 1
 PARAMETER C_S_AXIS_S2MM_TDATA_WIDTH = 16
 PARAMETER C_M_AXIS_MM2S_TDATA_WIDTH = 16
 PARAMETER C_NUM_FSTORES = 1
 PARAMETER C_INCLUDE_SG = 0
 PARAMETER C_INCLUDE_MM2S_DRE = 1
 PARAMETER C_INCLUDE_S2MM_DRE = 1
 PARAMETER C_INCLUDE_MM2S_SF = 1
 PARAMETER C_INCLUDE_S2MM_SF = 1
 PARAMETER C_MM2S_GENLOCK_NUM_MASTERS = 1
 PARAMETER C_MM2S_LINEBUFFER_THRESH = 2560
 PARAMETER C_FLUSH_ON_FSYNC = 1
 BUS_INTERFACE S_AXI_LITE = axi4lite_0
 BUS_INTERFACE M_AXI_MM2S = axi4_0
 BUS_INTERFACE M_AXI_S2MM = axi4_0
 BUS_INTERFACE M_AXIS_MM2S = axi_vdma_2_M_AXIS_MM2S
 BUS_INTERFACE S_AXIS_S2MM = axi_manr_0_M_AXIS_MEM
 PORT s_axi_lite_aclk = clk_50_0000MHzMMCM0
 PORT mm2s_fsync = axi_vtc_1_fsync
 PORT s2mm_fsync = axi_vtc_1_fsync
PORT mm2s_introut = axi_vdma_2_mm2s_introut
 PORT s2mm_introut = axi_vdma_2_s2mm_introut
 PORT m_axi_mm2s_aclk = clk_200_0000MHzMMCM0
 PORT m_axi_s2mm_aclk = clk_200_0000MHzMMCM0
 PORT s_axis_s2mm_aclk = clk_75_0000MHzMMCM0
 PORT m_axis_mm2s_aclk = clk_75_0000MHzMMCM0
END


BEGIN axi_scaler
 PARAMETER INSTANCE = axi_scaler_0
 PARAMETER HW_VER = 5.00.a
 PARAMETER C_SEPARATE_YC_COEFS = 1
 PARAMETER C_MAX_SAMPLES_OUT_PER_LINE = 1280
 PARAMETER C_MAX_PHASES = 64
 PARAMETER C_INIT_COEF_SOURCE = 1
 PARAMETER C_YC_FILTER_CONFIG = 1
 PARAMETER C_NUMBER_OF_H_TAPS = 11
```

```
 PARAMETER C_NUMBER_OF_V_TAPS = 11
 PARAMETER C_MAX_COEF_SETS = 16
 PARAMETER C_S_AXIS_TDATA_WIDTH = 32
 PARAMETER C_M_AXIS_TDATA_WIDTH = 16
 PARAMETER C_BASEADDR = 0x7c800000
 PARAMETER C_HIGHADDR = 0x7c80ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE S_AXIS = axi_manr_0_M_AXIS_OUTPUT
 BUS_INTERFACE M_AXIS = axi_scaler_0_M_AXIS
 PORT S_AXI_ACLK = clk_50_0000MHzMMCM0
 PORT video_in_clk = clk_75_0000MHzMMCM0
 PORT video_out_clk = clk_75_0000MHzMMCM0
 PORT clk = clk_150_0000MHzMMCM0
 PORT vsync_i = axi_vtc_1_fsync
 PORT IP2INTC_Irpt = axi_scaler_0_IP2INTC_Irpt
END


BEGIN axi_vdma
 PARAMETER INSTANCE = axi_vdma_1
 PARAMETER HW_VER = 3.01.a
 PARAMETER C_USE_FSYNC = 1
 PARAMETER C_MM2S_GENLOCK_MODE = 1
 PARAMETER C_MM2S_LINEBUFFER_DEPTH = 4096
 PARAMETER C_S2MM_LINEBUFFER_DEPTH = 4096
 PARAMETER C_MM2S_MAX_BURST_LENGTH = 256
 PARAMETER C_S2MM_MAX_BURST_LENGTH = 256
 PARAMETER C_MM2S_LINEBUFFER_THRESH = 2560
 PARAMETER C_PRMRY_IS_ACLK_ASYNC = 1
 PARAMETER C_S_AXIS_S2MM_TDATA_WIDTH = 16
 PARAMETER C_M_AXIS_MM2S_TDATA_WIDTH = 16
 PARAMETER C_NUM_FSTORES = 5
 PARAMETER C_INCLUDE_SG = 0
 PARAMETER C_INCLUDE_MM2S_DRE = 1
 PARAMETER C_INCLUDE_S2MM_DRE = 1
 PARAMETER C_INCLUDE_MM2S_SF = 1
 PARAMETER C_INCLUDE_S2MM_SF = 1
 PARAMETER C_MM2S_GENLOCK_NUM_MASTERS = 2
 PARAMETER C_S2MM_LINEBUFFER_THRESH = 2560
 PARAMETER C_BASEADDR = 0x7e200000
 PARAMETER C_HIGHADDR = 0x7e20ffff
 BUS_INTERFACE S_AXI_LITE = axi4lite_0
 BUS_INTERFACE M_AXI_MM2S = axi4_0
 BUS_INTERFACE M_AXI_S2MM = axi4_0
 BUS_INTERFACE M_AXIS_MM2S = axi_vdma_1_M_AXIS_MM2S
 BUS_INTERFACE S_AXIS_S2MM = axi_scaler_0_M_AXIS
 PORT s_axi_lite_aclk = clk_50_0000MHzMMCM0
 PORT m_axi_mm2s_aclk = clk_75_0000MHzMMCM0
 PORT m_axi_s2mm_aclk = clk_75_0000MHzMMCM0
 PORT mm2s_fsync = axi_vtc_1_fsync
 PORT s2mm_fsync = axi_vtc_1_fsync
 PORT s2mm_frame_ptr_out = axi_vdma_1_s2mm_frame_ptr_out
 PORT mm2s_frame_ptr_in = axi_vdma_0_s2mm_frame_ptr_out &
axi_vdma_1_s2mm_frame_ptr_out
 PORT mm2s_introut = axi_vdma_1_mm2s_introut
 PORT s2mm_introut = axi_vdma_1_s2mm_introut
 PORT s2mm_buffer_full = axi_vdma_1_s2mm_buffer_full
 PORT s2mm_buffer_almost_full = axi_vdma_1_s2mm_buffer_almost_full
END
```

```
BEGIN axi_osd
 PARAMETER INSTANCE = osd_0
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_LAYER1_TYPE = 1
 PARAMETER C_NUM_LAYERS = 2
 PARAMETER C_LAYER1_IMEM_SIZE = 96
 PARAMETER C_NUM_DATA_CHANNELS = 2
 PARAMETER C_ALPHA_CHANNEL_EN = 0
 PARAMETER C_OUTPUT_MODE = 1
 PARAMETER C_BASEADDR = 0x73a00000
 PARAMETER C_HIGHADDR = 0x73a0ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE S0_AXIS = axi_vdma_1_M_AXIS_MM2S
 BUS_INTERFACE XSVI_IN = axi_vtc_1_XSVI_OUT
 BUS_INTERFACE XSVI_OUT = osd_0_XSVI_OUT
 PORT clk = clk_75_0000MHzMMCM0
 PORT IP2INTC_Irpt = osd_0_IP2INTC_Irpt
END

BEGIN axi_interconnect
 PARAMETER INSTANCE = axi4lite_0
 PARAMETER HW_VER = 1.03.a
 PARAMETER C_INTERCONNECT_CONNECTIVITY_MODE = 0
 PORT INTERCONNECT_ARESETN = proc_sys_reset_0_Interconnect_aresetn
 PORT INTERCONNECT_ACLK = clk_50_0000MHzMMCM0
END

BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER HW_VER = 8.20.a
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_FPU = 0
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_ICACHE_BASEADDR = 0xc0000000
 PARAMETER C_ICACHE_HIGHADDR = 0xcfffffff
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_ICACHE_ALWAYS_USED = 1
 PARAMETER C_DCACHE_BASEADDR = 0xc0000000
 PARAMETER C_DCACHE_HIGHADDR = 0xcfffffff
 PARAMETER C_USE_DCACHE = 1
 PARAMETER C_DCACHE_ALWAYS_USED = 1
 PARAMETER C_INTERCONNECT_M_AXI_DC_AW_REGISTER = 0
 PARAMETER C_INTERCONNECT_M_AXI_DC_W_REGISTER = 0
 PARAMETER C_M_AXI_D_BUS_EXCEPTION = 1
 PARAMETER C_M_AXI_I_BUS_EXCEPTION = 1
 PARAMETER C_ILL_OPCODE_EXCEPTION = 1
 PARAMETER C_UNALIGNED_EXCEPTIONS = 1
 PARAMETER C_OPCODE_0x0_ILLEGAL = 1
 PARAMETER C_USE_STACK_PROTECTION = 1
 BUS_INTERFACE M_AXI_DP = axi4lite_0
 BUS_INTERFACE M_AXI_DC = axi4_0
 BUS_INTERFACE M_AXI_IC = axi4_0
 BUS_INTERFACE DEBUG = microblaze_0_debug
 BUS_INTERFACE DLMB = microblaze_0_dlmb
 BUS_INTERFACE ILMB = microblaze_0_ilmb
 PORT MB_RESET = proc_sys_reset_0_MB_Reset
 PORT CLK = clk_100_0000MHzMMCM0
```

```
 PORT INTERRUPT = microblaze_0_interrupt
END

BEGIN lmb_v10
 PARAMETER INSTANCE = microblaze_0_ilmb
 PARAMETER HW_VER = 2.00.b
 PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
 PORT LMB_CLK = clk_100_0000MHzMMCM0
END

BEGIN lmb_v10
 PARAMETER INSTANCE = microblaze_0_dlmb
 PARAMETER HW_VER = 2.00.b
 PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
 PORT LMB_CLK = clk_100_0000MHzMMCM0
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = microblaze_0_i_bram_ctrl
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0000ffff
 BUS_INTERFACE SLMB = microblaze_0_ilmb
 BUS_INTERFACE BRAM_PORT =
microblaze_0_i_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = microblaze_0_d_bram_ctrl
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0000ffff
 BUS_INTERFACE SLMB = microblaze_0_dlmb
 BUS_INTERFACE BRAM_PORT =
microblaze_0_d_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN bram_block
 PARAMETER INSTANCE = microblaze_0_bram_block
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA =
microblaze_0_i_bram_ctrl_2_microblaze_0_bram_block
 BUS_INTERFACE PORTB =
microblaze_0_d_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT Ext_Reset_In = RESET
 PORT MB_Reset = proc_sys_reset_0_MB_Reset
 PORT Slowest_sync_clk = clk_50_0000MHzMMCM0
 PORT Interconnect_aresetn = proc_sys_reset_0_Interconnect_aresetn
 PORT Dcm_locked = proc_sys_reset_0_Dcm_locked
 PORT MB_Debug_Sys_Rst = proc_sys_reset_0_MB_Debug_Sys_Rst
 PORT BUS_STRUCT_RESET = proc_sys_reset_0_BUS_STRUCT_RESET
END
```

```
BEGIN axi_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x83000000
 PARAMETER C_HIGHADDR = 0x8300ffff
 PARAMETER C_INTERCONNECT_S_AXI_MASTERS = plbv46_axi_bridge_0.M_AXI
 BUS_INTERFACE S_AXI = axi_interconnect_0
 PORT RX = fpga_0_RS232_Uart_1_RX_pin
 PORT TX = fpga_0_RS232_Uart_1_TX_pin
 PORT Interrupt = RS232_Uart_1_Interrupt
 PORT S_AXI_ACLK = clk_100_0000MHzMMCM0
END

BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
 PARAMETER HW_VER = 4.02.a
 PARAMETER C_CLKIN_FREQ = 200000000
 PARAMETER C_CLKOUT0_FREQ = 150000000
 PARAMETER C_CLKOUT0_GROUP = MMCM0
 PARAMETER C_CLKOUT1_FREQ = 200000000
 PARAMETER C_CLKOUT1_GROUP = MMCM0
 PARAMETER C_CLKOUT2_FREQ = 400000000
 PARAMETER C_CLKOUT2_GROUP = MMCM0
 PARAMETER C_CLKOUT3_FREQ = 400000000
 PARAMETER C_CLKOUT3_GROUP = MMCM0
 PARAMETER C_CLKOUT3_BUF = FALSE
 PARAMETER C_CLKOUT3_VARIABLE_PHASE = TRUE
 PARAMETER C_CLKOUT4_FREQ = 50000000
 PARAMETER C_CLKOUT4_GROUP = MMCM0
 PARAMETER C_CLKOUT5_FREQ = 75000000
 PARAMETER C_CLKOUT5_GROUP = MMCM0
 PARAMETER C_CLKOUT6_FREQ = 150000000
 PARAMETER C_CLKOUT6_GROUP = MMCM0
 PORT RST = RESET
 PORT CLKIN = CLK
 PORT CLKOUT0 = clk_100_0000MHzMMCM0
 PORT CLKOUT1 = clk_200_0000MHzMMCM0
 PORT CLKOUT2 = clk_400_0000MHzMMCM0
 PORT CLKOUT3 = clk_400_0000MHzMMCM0_nobuf_varphase
 PORT CLKOUT4 = clk_50_0000MHzMMCM0
 PORT CLKOUT5 = clk_75_0000MHzMMCM0
 PORT CLKOUT6 = clk_150_0000MHzMMCM0
 PORT LOCKED = proc_sys_reset_0_Dcm_locked
 PORT PSCLK = clk_50_0000MHzMMCM0
 PORT PSEN = psen
 PORT PSINCDEC = psincdec
 PORT PSDONE = psdone
END

BEGIN mdm
 PARAMETER INSTANCE = debug_module
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_UART = 1
 PARAMETER C_BASEADDR = 0x74800000
```

```
 PARAMETER C_HIGHADDR = 0x7480ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE MBDEBUG_0 = microblaze_0_debug
 PORT S_AXI_ACLK = clk_50_0000MHzMMCM0
 PORT Debug_SYS_Rst = proc_sys_reset_0_MB_Debug_Sys_Rst
END

BEGIN axi_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 1
 PARAMETER C_BASEADDR = 0x40600000
 PARAMETER C_HIGHADDR = 0x4060ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT TX = RS232_Uart_1_sout
 PORT RX = RS232_Uart_1_sin
 PORT S_AXI_ACLK = clk_50_0000MHzMMCM0
END

BEGIN axi_v6_ddrx
 PARAMETER INSTANCE = DDR3_SDRAM
 PARAMETER HW_VER = 1.03.a
 PARAMETER C_MEM_PARTNO = MT4JSF6464HY-1G1
 PARAMETER C_INTERCONNECT_S_AXI_AR_REGISTER = 1
 PARAMETER C_INTERCONNECT_S_AXI_AW_REGISTER = 1
 PARAMETER C_INTERCONNECT_S_AXI_R_REGISTER = 1
 PARAMETER C_INTERCONNECT_S_AXI_W_REGISTER = 1
 PARAMETER C_INTERCONNECT_S_AXI_B_REGISTER = 1
 PARAMETER C_INTERCONNECT_S_AXI_READ_ACCEPTANCE = 8
 PARAMETER C_INTERCONNECT_S_AXI_WRITE_ACCEPTANCE = 8
 PARAMETER C_S_AXI_DATA_WIDTH = 128
 PARAMETER C_INTERCONNECT_S_AXI_MASTERS = microblaze_0.M_AXI_DC &
microblaze_0.M_AXI_IC & axi_vdma_0.M_AXI_MM2S & axi_vdma_0.M_AXI_S2MM &
axi_vdma_1.M_AXI_MM2S & axi_vdma_1.M_AXI_S2MM
 PARAMETER C_MMCM_EXT_LOC = MMCM_ADV_X0Y8
 PARAMETER C_S_AXI_BASEADDR = 0xc0000000
 PARAMETER C_S_AXI_HIGHADDR = 0xcfffffff
 BUS_INTERFACE S_AXI = axi4_0
 PORT ddr_ck_p = ddr_memory_clk
 PORT ddr_ck_n = ddr_memory_clk_n
 PORT ddr_cke = ddr_memory_cke
 PORT ddr_cs_n = ddr_memory_cs_n
 PORT ddr_odt = ddr_memory_odt
 PORT ddr_ras_n = ddr_memory_ras_n
 PORT ddr_cas_n = ddr_memory_cas_n
 PORT ddr_we_n = ddr_memory_we_n
 PORT ddr_dm = ddr_memory_dm
 PORT ddr_ba = ddr_memory_ba
 PORT ddr_addr = ddr_memory_addr
 PORT ddr_reset_n = ddr_memory_ddr3_rst
 PORT ddr_dq = ddr_memory_dq
 PORT ddr_dqs_p = ddr_memory_dqs
 PORT ddr_dqs_n = ddr_memory_dqs_n
 PORT clk = clk_200_0000MHzMMCM0
 PORT clk_ref = clk_200_0000MHzMMCM0
 PORT clk_mem = clk_400_0000MHzMMCM0
```

```
 PORT clk_rd_base = clk_400_0000MHzMMCM0_nobuf_varphase
 PORT PD_PSEN = psen
 PORT PD_PSINCDEC = psincdec
 PORT PD_PSDONE = psdone
END

BEGIN axi_intc
 PARAMETER INSTANCE = microblaze_0_intc
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x41200000
 PARAMETER C_HIGHADDR = 0x4120ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT IRQ = microblaze_0_interrupt
 PORT S_AXI_ACLK = clk_50_0000MHzMMCM0
 PORT INTR = osd_0_IP2INTC_Irpt & axi_vdma_0_mm2s_introut &
axi_vdma_0_s2mm_introut & timebase_1_IP2INTC_Irpt &
axi_scaler_0_IP2INTC_Irpt & axi_vdma_1_mm2s_introut &
axi_vdma_1_s2mm_introut
END
```

XILINX®

Appendix A

Verification, Compliance, and Interoperability

This appendix includes details on simulation and testing.

Simulation

A parameterizable test bench was used to test the MANR core. Testing included the following:

- Register accessing
- Processing of multiple frames of data
- Various frame sizes
- Various MANR strengths
- Various AXI4-Stream data bus widths.

Hardware Testing

The MANR core has been tested in a variety of hardware platforms at Xilinx for various parameterizations, including the following:

- A test design was developed for the core that incorporated a MicroBlaze processor, AXI4 interface and various other peripherals, as described in Chapter 6, Detailed Example Design.
- The software for the test system included input frames embedded in the source-code. The checksums of the processed images were also pre-calculated and included in the software. The frames, resident in external memory, are read by the AXI_VDMA, processed by the MANR and the result is passed back to memory. Software then accesses the processed frame in memory and calculates its checksum. This matches the pre-calculated checksum.
- Various configurations were implemented in this way. The C model was used to create the expected checksums and generate the stimulus C code frame data that is compiled into the software.
- Pass/fail status is reported by the software.

In addition, the MANR core has been more regularly tested using an automated validation flow. Primarily, this instantiates the core in order to read registers back, validating the core's Version register and proving that it has been implemented in the design. This has been run regularly in order to validate new core versions during development, and also to guard against EDK tools regressions.

XILINX®

Appendix B

Migrating

This appendix describes migrating from older versions of the IP to the current IP release.

Migrating to the EDK pCore AXI4-Lite Interface

The MANR v2.0 changed from the PLB processor interface to the EDK pCore AXI4-Lite interface. As a result, all of the PLB-related connections have been replaced with an AXI4-Lite interface. For more information, see UG761, *AXI Reference Guide.*

Migrating to the AXI4-Stream Interface

The MANR v3.0 changed from the Video Frame Buffer Controller (VFBC) native interfaces to the AXI4-Stream interfaces. As a result, all of the VFBC-related connections have been replaced with AXI4-Stream connections. For more information, see UG761, *AXI Reference Guide.*

Parameter Changes in the XCO File

For the MANR v3.0 core, the XCO file now includes AXI4-Stream IO data widths:

- s_axis_currframe_tdata_width
- s_axis_prevframe_tdata_width
- m_axis_mem_tdata_width
- m_axis_output_tdata_width

Port Changes

The MANR v3.0 core now includes:

- AXI4-Stream Input interface signals for Current frame, `s_axis_currframe`
- AXI4-Stream Input interface signals for Previous frame, `s_axis_prevframe`
- AXI4-Stream Output interface signals for temporal feedback path, `m_axis_mem`
- AXI4-Stream Output interface signals for temporal downstream, `m_axis_output`

The MANR v3.0 core no longer includes:

- VFBC Input signals for current and previous frames interleaved
- VFBC Output signals for temporal feedback path
- Any VFBC command buses
- YCM Output signals

- The following Control inputs/pCore registers:
  - Current, previous and output frame start addresses
  - VFBC Stride
  - Chroma valid output

Functionality Changes

There were primarily two functionality changes for the MANR v3.0 core.

Interface Changes

In the MANR v2.0 core, the current and previous frames were provided into the MANR core via a common VFBC interface, interleaved on a line basis: Currframe Line0, Prevframe Line0, Currframe Line1, Prevframe Line1, Currframe Line2, Prevframe Line2. This also necessitated the use of approximately a double pixel-rate clock frequency in order to achieve sufficient bandwidth from that VFBC.

In order to achieve 1080P60 throughput in Spartan-6 devices, the MANR v3.0 core has been adapted to accept current and previous frames through separate interfaces (now using AXI4-Stream instead of VFBC). The clock frequency required by the MANR is now only 1 x pixel-rate.

Independent Chroma-Channel Processing

In the MANR v2.0 core, motion calculations were made only on the Luma channel. Decisions were made on the Luma channel according to these calculations. The calculations were also subsampled and applied to the 4:2:*x* Chroma channel. This led to a substandard response on the Chroma channel.

In the MANR v3.0 core, the Chroma channel motion calculations are made independently of the Luma channel, adding to the resource count by a small margin.

XILINX®

Appendix C

Debugging

Some general debugging tips are as follows:

- Verify the Version register can be read properly. See Table 2-5 for register definitions.
- Verify the other resisters can be read properly. Do they match your expectations?
- Verify that bits 0 and 1 of the core's Control register are both set to “1”. Bit 0 is the Core Enable bit. Bit 1 is the Register Update Enable bit.
- Verify that the `vsync_in` input is being properly driven.
- Verify that the output interface is not holding off permanently. The `m_axis_xxx_tready` signal must be High for any data to come out of the core.
- Verify that the MANR core is toggling its `m_axis_xxx_tvalid` outputs. If this is occurring, check the data output.
- When the downstream output `m_axis_output` interface is *not* in use, ensure that the tReady signal is not driven Low.
- Check that the addresses for the MANR input/output frame buffers are correct.
- Use XMD (or other utility) to check the actual data in memory before and/or after processing in the MANR core.
- Attach a static pattern-generator in order to introduce known data into the video stream.

XILINX®

Appendix D

Application Software Development

This appendix details the use of the software drivers that are included with the pCore version of the MANR core.

Device Drivers

pCore Driver Files

The MANR 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 MANR core. A low-level API is also provided for developers to access the core directly through the system registers described in EDK pCore Files in Chapter 3. Table 3-3, page 23 lists the files included with the MANR pCore driver.

pCore API Functions

This section describes the functions included for the pCore driver generated for the MANR pCore. To use the API functions provided, the following header files must be included in the user's C code:

```
#include “xparameters.h”
#include “xmanr.h”
```

The system hardware settings, including the base address of the MANR core, are defined in the `xparameters.h` file. The `xmanr.h` file provides the API access to all of the features of the MANR device driver.

More detailed documentation of the API functions can be found by opening `index.html` in the pCore directory `manr_v1_01_a/doc/html/api`.

Functions in xmanr.c

- int XMANR_CfgInitialize (XMANR *InstancePtr, XMANR_Config *CfgPtr, u32 EffectiveAddr)

  This function initializes a MANR core.

- void XMANR_SetFrameSize (XMANR *InstancePtr, u32 Height, u32 Width, u32 Stride)

  This function sets up the frame size information used by a MANR device. Note that 'stride' parameter is now obsolete and set to 0.

- void XMANR_GetFrameSize (XMANR *InstancePtr, u32 *HeightPtr, u32 *WidthPtr, u32 *StridePtr)

  This function sets up the frame size information used by a MANR device. Note that 'StridePtr' is now obsolete.

- void XMANR_LoadMtfBank(XMANR *InstancePtr, u8 BankIndex, u32 *MTFData)

  This function loads the Motion Transfer LUT configuration to be used by a MANR device.

- void XMANR_GetVersion(XMANR *InstancePtr, u16 *Major, u16 *Minor, u16 *Revision)

  This function returns the version of a MANR device.

Functions in xmanr_sinit.c

- XMANR_Config * XMANR_LookupConfig (u16 DeviceId)

  XMANR_LookupConfig returns a reference to a XMANR_Config structure based on the unique device ID, DeviceId.

Functions in xmanr_intr.c

- void XMANR_IntrHandler (void *InstancePtr)

  This function is the interrupt handler for the MANR driver.

- int XMANR_SetCallBack (XMANR *InstancePtr, u32 HandlerType, void *CallBackFunc, void *CallBackRef)

  This routine installs an asynchronous callback function for the given HandlerType.

XILINX®

Appendix E

C Model Reference

This appendix introduces the bit accurate C model for the Motion Adaptive Noise Reduction v3.0 core, which has been developed primarily for system-level modeling. Features of this C model include:

- Bit accurate with v_manr_v3_0 core
- Library module for the MANR core function
- Available for 32 and 64-bit Windows and 32 and 64-bit Linux platforms
- Supports all features of the HW core that affect numerical results
- Designed for rapid integration into a larger system model
- Example application C code is provided to show how to use the function

Overview

The bit accurate C model for the LogiCORE IP MANR v3.0 can be used on 32/64-bit Windows and 32/64-bit Linux platforms. The model comprises a set of C functions, which reside in a statically linked library (shared library). Full details of the interface to these functions are provided in Interface, page 58.

The main features of the C model package are:

- **Bit Accurate C Model** - produces the same output data as the MANR v3.0 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.
- **Application Source Code** - uses the model library function. This can be used as example code showing how to use the library function. However, it also serves these purposes:
  - **Input .yuv file** is processed by the application; 8-bit YUV422 format accepted.
  - **Output .yuv file** is generated by the application; 8-bit YUV422 format generated.
  - **Report.txt file** is generated for run time status and error messages.

The latest version of the model is available for download on the LogiCORE IP MANR product page at: http://www.xilinx.com/products/ipcenter/EF-DI-IMG-MA-NOISE.htm

Unpacking and Model Contents

Unzip the `v_manr_v3_0_bitacc_model` file, containing the bit accurate models for the MANR IP Core. This creates the directory structure and files in Table E-1

Table E-1: **Directory Structure and Files of MANR C Model**

File Name	Contents
./doc	Documentation directory
README.txt	release notes
<product Guide>.pdf	This document.
Makefile	Makefile for running gcc via make for 32-bit and 64-bit Linux platforms
v_manr_v3_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 test bench stimulus file I/O
run_bitacc_cmodel.c	example code calling the C model
run_bitacc_cmodel_config.c	example code calling the C model – uses command line and config file arguments
run_bitacc_cmodel.sh	Bash shell script that compiles and runs the model.
./lin64	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms.
libIp_v_manr_v3_0_bitacc_cmodel.so	model shared object library
libIp_v_utils_v1_0_bitacc_cmodel.so	Utilities shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_manr_v3_0_bitacc_cmodel.so
run_bitacc_cmodel	64-bit Windows fixed configuration executable
./lin	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms.
libIp_v_manr_v3_0_bitacc_cmodel.so	model shared object library
libIp_v_utils_v1_0_bitacc_cmodel.so	Utilities shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_manr_v3_0_bitacc_cmodel.so

Table E-1: **Directory Structure and Files of MANR C Model**

File Name	Contents
run_bitacc_cmodel	32-bit Windows fixed configuration executable
./nt64	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms.
libIp_v_manr_v3_0_bitacc_cmodel.dll	Precompiled library file for 64-bit Windows platforms compilation
libIp_v_manr_v3_0_bitacc_cmodel.lib	Precompiled library file for 64-bit Windows platforms compilation
libIp_v_utils_v1_0_bitacc_cmodel.dll	Precompiled utilities library file for 64-bit Windows platforms compilation
libIp_v_utils_v1_0_bitacc_cmodel.lib	Precompiled utilities library file for 64-bit Windows platforms compilation
stlport.5.1.dll	STL library
run_bitacc_cmodel.exe	64-bit Windows fixed configuration executable
./nt	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms.
libIp_v_manr_v3_0_bitacc_cmodel.dll	Precompiled library file for 32-bit Windows platforms compilation
libIp_v_manr_v3_0_bitacc_cmodel.lib	Precompiled library file for 32-bit Windows platforms compilation
libIp_v_utils_v1_0_bitacc_cmodel.dll	Precompiled utilities library file for 32-bit Windows platforms compilation
libIp_v_utils_v1_0_bitacc_cmodel.lib	Precompiled utilities library file for 32-bit Windows platforms compilation
stlport.5.1.dll	STL library
run_bitacc_cmodel.exe	32-bit Windows fixed configuration executable
./examples	
video_in.yuv	Example YUV input file, resolution 1280Hx720V
video_in.hdr	Header file for video_in.yuv
video_in_128x128.yuv	Example YUV input file, resolution 128Hx128V
video_in_128x128.hdr	Header file for video_in_128x128.yuv

Software Requirements

The MANR v3.0 C models were compiled and tested with the software listed in Table E-2.

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

Platform	C Compiler
32/64-bit Linux	GCC 4.1.1
32/64-bit Windows	Microsoft Visual Studio 2005

Interface

The MANR core function is a statically linked library. A higher level software project can make function calls to this function:

```
int xilinx_ip_v_manr_v3_v3_0_bitacc_simulate(
      struct xilinx_ip_v_manr_v3_0_generics generics,
      struct xilinx_ip_v_manr_v3_0_inputs inputs,
      struct xilinx_ip_v_manr_v3_0_outputs*  outputs).
```

Before using the model, the structures holding the inputs, generics and output of the MANR instance must be defined:

```
struct  xilinx_ip_v_manr_v3_0_generics manr_generics;
struct  xilinx_ip_v_manr_v3_0_inputs  manr_inputs;
struct xilinx_ip_v_manr_v3_0_outputs* manr_outputs
```

The declaration of these structures are in the `v_manr_v3_0_bitacc_cmodel.h` file.

Before making the function call, complete these steps:

1. Populate the *generics* structure:

   nr_strength - Between 0 and 4. Describes the strength of the initial noise reduction filter: 0= None; 1=Weak; 2=Med; 3=Strong; 4=Aggressive.

2. Populate the *inputs* structure to define the values of run time parameters:

   Note: This function processes *one frame at a time*.

   - **video_in** - Video structure that comprises these elements:
     - **bits_per_component** - Must be set to 8.
     - **cols** - Horizontal image size: 32 to 1920.
     - **rows** - Vertical image size: 32 to 1080.
     - **frames** - Set to 1; this function processes *one frame at a time.*
     - **mode** - Defines the chroma format (RGB, YUV422, and so on); see Table E-4. This core can only process YC422 or YC420.
     - **data** - This is the frame of video data to be processed, arranged in raster form.
   - **mtf** - MTF Look-up table. This is a 1D array of 64 integers in the range 0 to 255, which represents the Motion Transfer Function.

3. Populate the *outputs* structure.
   - **video_out** - Video structure that comprises the same elements as the video_in structure element described previously.

Note: The `video_in` variable is not initialized because the initialization depends on the actual test image to be simulated. The next section describes the initialization of the `video_in` structure.

Results are provided in the outputs structure, which contains the output video data in the form of type `video_struct`. After the outputs have been evaluated or saved, dynamically allocated memory for input and output video structures must be released. See Delete the Video Structure, page 60 for more information. Successful execution of all provided functions return a value of 0. Otherwise, a non-zero error code indicates that problems were encountered during function calls.

Input and Output Video Structure

Input images or video streams can be provided to the MANR v3.0 reference model using the general purpose `video_struct` structure, defined in `video_utils.h`:

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

Table E-3: **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(1)
Cols	Number of columns per line(1)
Bit_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 E-4.
Data	Set of five pointers to three dimensional arrays containing data for image planes. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col]. In the MANR C model case, only one frame is processed at any one time. Consequently, the '[frame]' index is always set to 0.

1. Pertaining to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through all frames of the video stream, however, different planes, such as Y,U and V can have different dimensions.

Table E-4: **Named Constants for Video Modes With Corresponding Planes and Representations**

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(1)	3	422 format YUV video, (U,V chrominance channels horizontally sub-sampled)
FORMAT_C420(1)	3	420 format YUV video, (U,V sub-sampled both horizontally and vertically)

Table E-4: **Named Constants for Video Modes With Corresponding Planes and Representations**

FORMAT_MONO_M	3	Monochrome (luminance) video with motion
FORMAT_RGBA	4	RGB image/video data with alpha (transparency) channel
FORMAT_C420_M	5	420 YUV video with motion
FORMAT_C422_M	5	422 YUV video with motion
FORMAT_C444_M	5	444 YUV video with motion
FORMAT_RGBM	5	RGB video with motion

1. Supported by the MANR core.

Working With Video_struct Containers

The header file `video_utils.h` 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);
```

The function `video_planes_per_mode` returns the number of component planes defined by the mode variable, as described in Table E-4. The 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`:

```
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.

The following example 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 as follows:

free_video_buff ((struct video_struct*) &manr_outputs.video_out);

Example Code

An example C file, `run_bitacc_cmodel.c`, is provided along with the lin32, lin64, NT32 and NT64 executables for this example. This C file has these characteristics:

- Contains an example of how to write an application that makes a function call to the MANR 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 for use by the model.
- Uses a YUV file writing function to provide an optional output YUV file, which allows the user to visualize the result of the MANR operation.

The delivered model extracts a number of frames from the specified .yuv input file, removes noise from this video stream, and outputs the noise reduced stream in the specified .yuv output file.

The MANR algorithm is temporally recursive. Motion is determined by comparing the current frame with the previous frame. For the first input frame, there is no previous frame, so the first output frame always shows zero motion.

The MTF (motion transfer function) determines the level to which each of the two frames contributes to the output frame. The nr_strength parameter selects between five different MTF characteristics. These functions are coded into the wrapper function `run_bitacc_cmodel.c`.

Initializing the MANR Input Video Structure

In the example code wrapper, data is assigned to a video structure by reading from a .yuv video file. This file is described in C Model Example I/O Files, page 62. The `yuv_utils.h` and `video_utils.h` header files packaged with the bit accurate C models contain functions to facilitate file I/O. The `run_bitacc_cmodel` example code uses these functions to read from the YUV file.

YUV Image Files

The header `yuv_utils.h` file declares functions that help access files in standard YUV format. It operates on images with three planes (Y, U and V). The following functions operate on arguments of type yuv8_video_struct, which is defined in `yuv_utils.h`.

```
int write_yuv8(FILE *outfile, struct yuv8_video_struct *yuv8_video);
int read_yuv8(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 );
```

Note: All image/video manipulation utility functions expect both input and output structures to be initialized. For example, pointing to a structure to which memory has been allocated, either as static or dynamic variables. Moreover, the input structure must have the dynamically allocated container (data or y ,u, v) structures 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 generate 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.

C Model Example I/O Files

Input Files

- **<input_filename>.yuv** (Optional; for example, `video_in.yuv`, `video_in_128x128.yuv`).
  - Standard 8-bit YUV file format. Entire Y plane followed by entire Cb plane, followed by the entire Cr plane.
  - Can be viewed in a YUV player.
  - No header.

Output Files

- **<output_filename>.yuv** (Optional; for example, `video_out.yuv`).
  - Standard 8-bit 4:2:2 yuv file format. Entire Y plane followed by entire Cb plane, followed by the entire Cr plane.
  - Can be viewed in a YUV player.

Compiling the MANR v3.0 C Model With Example Wrapper

Linux (32/64 bits)

For 64-bit Linux, cd into the /lin64 directory. From there, run the command:

```
gcc -m64 -x c++ ../run_bitacc_cmodel.c -o run_bitacc_cmodel -L.
-lIp_v_manr_v3_0_bitacc_cmodel -Wl,-rpath,.
```

When using 32-bit Linux, cd into the /lin directory, and run with the '-m32' switch:

```
gcc -m32 -x c++ ../run_bitacc_cmodel.c -o run_bitacc_cmodel -L.
-lIp_v_manr_v3_0_bitacc_cmodel -Wl,-rpath,.
```

To run either 32- or 64-bit executables:

1. Set your LD_LIBRARY_PATH environment variable to the location of the two .so libraries.
2. Execute as follows:

```
./run_bitacc_cmodel video_in.yuv video_out.yuv 10 1280 720 2 1
```

Windows (32/64-bits)

The `v_manr_v3_0_bitacc_cmodel.zip` file includes all the4 necessary files required to compile the top-level demonstration code `run_bitacc_cmodel.c` with an ANSI C compliant compiler under Windows.

This section includes an example using Microsoft Visual Studio.

In Visual Studio, create a new, empty Win32 Console Application project. In the appropriate project folders, add the following files:

- `v_manr_v3_0_bitacc_cmodel.h`
- `libIp_v_manr_v3_0_bitacc_cmodel.lib`
- `libIp_v_utils_v1_0_bitacc_cmodel.lib`
- `run_bitacc_cmodel.c`

To run either 32- or 64-bit executables:

1. Cd to a location that includes all the following files
   - `run_bitacc_cmodel.exe` (or the executable file generated by your compiler)
   - `libIp_v_manr_v3_0_bitacc_cmodel.dll`
   - `libIp_v_utils_v1_0_bitacc_cmodel.dll`
2. Execute as follows:

```
run_bitacc_cmodel
<input_file>.yuv
<output_file>.yuv
<#frames>
<hsize>
<vsize>
<chroma_format>
<NR_strength>
```

For example:

```
run_bitacc_cmodel video_in.yuv video_out.yuv 10 1280 720 2 2
```

Compile/Run Shell Script

To compile the example code, use the cd command to 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. They are in the /lin64 directory. Use the cd command to go into the lin/lin64 directory and execute the bash shell script that compiles the project using the GNU C Compiler and runs it:

```
bash run_bitacc_cmodel.sh
```

The bash script text is provided here:

```
#!/bin/bash
################################################
# Compile model and libraries
################################################
gcc -x c++ ../run_bitacc_cmodel.c -o run_bitacc_cmodel -L. -lIp_v_manr_v3_0_bitacc_cmodel
-Wl,-rpath,.
```

```
################################################
# Run model.
# Usage:
#     ./run_bitacc_model <input_file>.yuv <output_file>.yuv <#frames> <hsize> <vsize>
<chroma_format> <NR_strength>
#     chroma_format: 1 = 4:2:0
#                    2 = 4:2:2
#     NR_strength:   0 = None
#                    1 = Weak
#                    2 = Medium
#                    3 = Strong
#                    4 = Aggressive
# Example:
# ./run_bitacc_cmodel ../video_in.yuv fred.yuv 10 1280 720 2 1
################################################
./run_bitacc_cmodel ../video_in.yuv video_out.yuv 10 1280 720 2 1
```

The user can customize this shell script.

XILINX®

Appendix F

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.

References

These documents provide supplemental material useful with this user guide:

- *AMBA® AXI4-Stream Protocol Specification*
- UG761, *AXI Reference Guide*
- DS768, *AXI Interconnect IP Data Sheet*

To search for Xilinx documentation, go to http://www.xilinx.com/support

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 LogiCORE IP Motion Adaptive Noise Reduction core is provided under the Xilinx End User 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 Motion Adaptive Noise Reduction core 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