LogiCORE IP Motion Adaptive Noise Reduction v4.00.a

Product Guide

PG006 April 24, 2012

XILINX®

XILINX®

Table of Contents

Appendix F: Additional Resources

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™ or EDK GUI, but is also programmable at runtime via the register interface.

Features

- Programmable register control.
- Optional AXI4-Lite Dynamic Control interface or fixed-mode operation.
- Selectable and programmable motion transfer function (MTF).
  - Five pre-loaded MTF curves: none, weak, medium, strong, and aggressive.
- 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 spatial resolutions from 32x32 to 4096x4096.
  - Supports 1080P60 in all supported device families

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Zynq™-7000, Artix™-7, Virtex®-7, Kintex®-7, Virtex-6, Spartan®-6
Supported User Interfaces	AXI4-Lite, AXI4-Stream
Resources	See Table 2-1 through Table 2-6.
Provided with Core	
Documentation	Product Guide
Design Files	NGC netlist, Encrypted HDL
Example Design	Not Provided
Test Bench	Verilog (3)
Constraints File	Not Provided
Simulation Models	VHDL or Verilog Structural, C-Model (3)
Tested Design Tools	
Design Entry Tools	CORE Generator™ tool, Platform Studio (XPS)
Simulation(4)	Mentor Graphics ModelSim, Xilinx ISim
Synthesis Tools	Xilinx Synthesis Technology (XST)
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.
2. Video Protocol as defined in UG761, *Xilinx AXI Reference Guide*.
3. HDL test bench and C-Model available on the product page on Xilinx.com at www.xilinx.com/products/intellectual-property/EF-DI-IMG-MA-NOISE.htm.
4. For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.

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.

Feature Summary

The MANR core supports resolutions of up to 4096x4096 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 temporal 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 EDK 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 Video protocol 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 Video protocol.

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.

Applications

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

Licensing and Ordering

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The core may be generated using either the Xilinx ISE CORE Generator tool or Embedded Edition software (EDK). For full access to all core functionality in simulation and in hardware, you must purchase a license for the core. Please contact your local Xilinx sales representative for information on pricing and availability of Xilinx LogiCORE IP.

For more information, please visit the LogiCORE IP Motion Adaptive Noise Reduction product page.

Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE modules and software, please contact your local Xilinx sales representative.

Product Specification

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-Stream Video Protocol and AXI4-Lite interconnect standards. See UG761, *Xilinx AXI Reference Guide*, for additional information.

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

This section contains typical clock frequencies for the target devices. These 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

For an accurate measure of the usage of primitives, slices, and CLBs for a particular instance, check the **Display Core Viewer after Generation** check box in the CORE Generator interface. To help guide the user making system-level and board-level decisions, Table 2-1 through Table 2-6 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.

Table 2-1: **Resources and Performance for Zynq-7000 Devices**

AXI4-Lite?	Max Video Resolution	Noise Reduction	FFs	LUTs	BRAM 36/18	DSP48	FMax (MHz) for each Speed-Grade		
							-1	-2	-3
No	128x128	0	808	930	0/1	3	232.5	247	328.5
No	1280x720	1	808	930	0/1	3	232.5	247	272.5
No	1920x1080	2	808	935	0/1	3	232.5	247	272.5
Yes	128x128	0	1242	1422	0/1	3	232.5	247	328.5

Table 2-2: **Resources and Performance for Artix-7 Devices**

AXI4-Lite?	Max Video Resolution	Noise Reduction	FFs	LUTs	BRAM 36/18	DSP48	FMax (MHz) for each Speed-Grade		
							-1	-2	-3
No	128x128	0	808	937	0/1	3	197.5	239	272.5
No	1280x720	1	808	938	0/1	3	197.5	239	272.5
No	1920x1080	2	808	935	0/1	3	197.5	239	272.5
Yes	128x128	0	1242	1351	0/1	3	197.5	239	272.5

Table 2-3: **Resources and Performance for Kintex-7 Devices**

AXI4-Lite?	Max Video Resolution	Noise Reduction	FFs	LUTs	BRAM 36/18	DSP48	FMax (MHz) for each Speed-Grade		
							-1	-2	-3
							-1	-2	-3
No	128x128	0	808	937	0/1	3	232.5	280	337.5
No	1280x720	1	808	943	0/1	3	232.5	280	337.5
No	1920x1080	2	808	953	0/1	3	232.5	280	337.5
Yes	128x128	0	1242	1436	0/1	3	232.5	280	337.5

Table 2-4: **Resources and Performance for Virtex-7 Devices**

AXI4-Lite?	Max Video Resolution	Noise Reduction	FFs	LUTs	BRAM 36/18	DSP48	FMax (MHz) for each Speed-Grade		
							-1	-2	-3
No	128x128	0	808	944	0/1	3	232.5	283	329
No	1280x720	1	808	947	0/1	3	232.5	283	329
No	1920x1080	2	808	952	0/1	3	232.5	283	329
Yes	128x128	0	1242	1451	0/1	3	232.5	283	329

Table 2-5: **Resources and Performance for Virtex-6 Devices**

AXI4-Lite?	Max Video Resolution	Noise Reduction	FFs	LUTs	BRAM 36/18	DSP48	FMax (MHz) for each Speed-Grade		
							-1	-2	-3
No	128x128	0	808	937	0/1	3	216	262	293.5
No	1280x720	1	808	938	0/1	3	216	262	293.5
No	1920x1080	2	808	937	0/1	3	216	262	293.5
Yes	128x128	0	1242	1334	0/1	3	216	262	293.5

Table 2-6: **Resources and Performance for Spartan-6 Devices**

AXI4-Lite?	Max Video Resolution	Noise Reduction	FFs	LUTs	BRAM 16/8	DSP48	FMax (MHz) for each Speed-Grade	
							-2	-3
No	128x128	0	808	916	0/1	3	175	195.5
No	1280x720	1	808	906	0/1	3	175	195.5
No	1920x1080	2	808	917	0/1	3	175	195.5
Yes	128x128	0	1244	1409	0/1	3	175	195.5

Port Descriptions

The MANR core uses industry standard control and data interfaces to connect to other system components. The following sections describe the various interfaces available with the core. Figure 2-1 illustrates an I/O diagram of the Video Scaler core. Some signals are optional and not present for all configurations of the core. The AXI4-Lite interface and the IRQ pin are present only when the core is configured through the GUI with an AXI4-Lite control interface. The INTC_IF interface is present only when the core is configured through the GUI with the INTC interface enabled.

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.

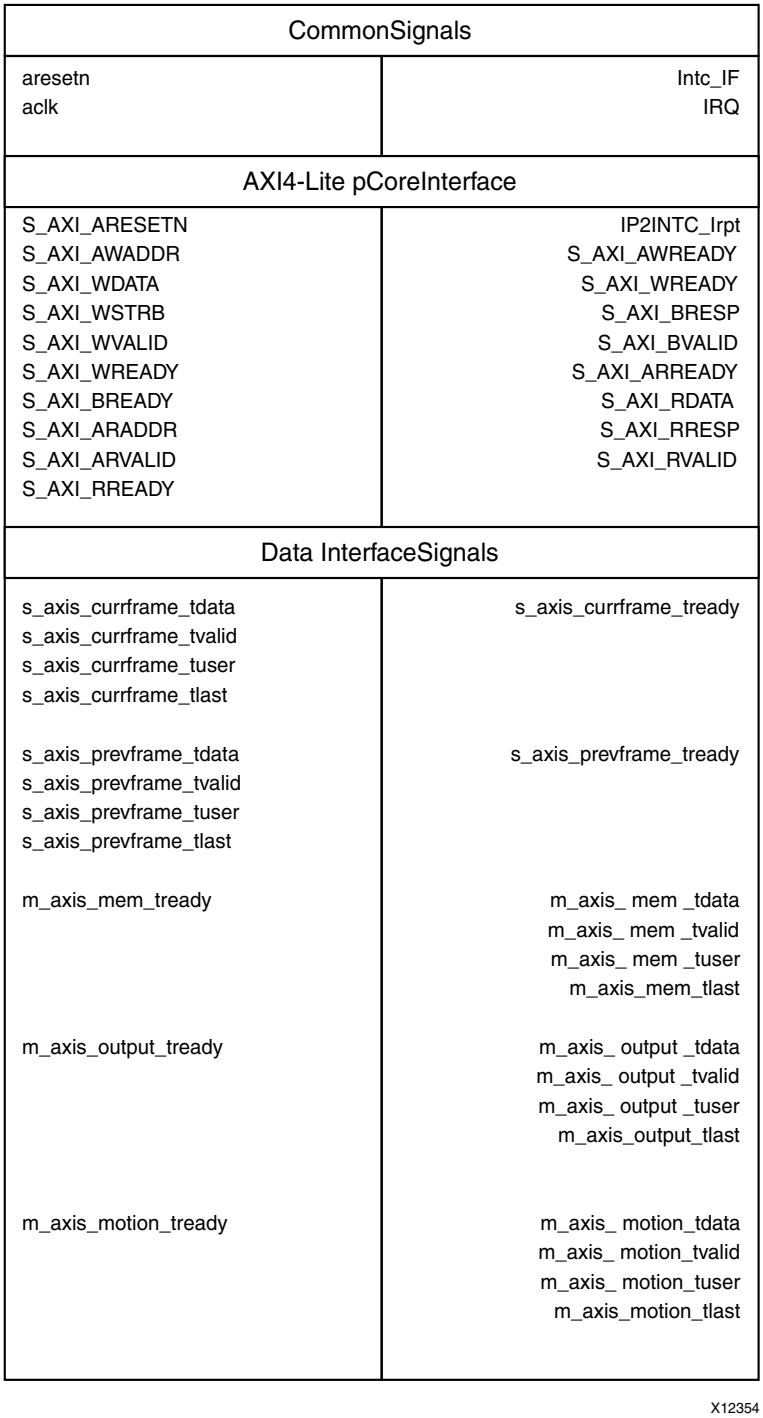

Figure 2-1: **MANR Interfaces and Signals**

Common Interface Signals

Table 2-7 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-7: **Common Interface Signals**

Signal Name	Direction	Width	Description
ACLK	In	1	Clock.
ARESETn	In	1	Active Low synchronous reset.
INTC_IF	Out	31	Optional external interrupt controller interface. Available only when INTC_IF is selected in the GUI.
IRQ	Out	1	Optional interrupt request pin. Available only when INTC_IF is selected in the GUI.

The ACLK and ARESETn signals are shared between the core, the AXI4-Stream data interfaces, and the AXI4-Lite control interface. See Interrupt Subsystem for a description of the INTC_IF and IRQ pins.

ACLK

All signals, including the AXI4-Stream and AXI4-Lite component interfaces, must be synchronous to the core clock signal ACLK. All interface input signals are sampled on the rising edge of ACLK. All output signal changes occur after the rising edge of ACLK.

ARESETn

The ARESETn pin is an active-low, synchronous reset input pertaining to both the AXI4-Stream and AXI4-Lite interfaces. When ARESETn is set to 0, the core resets at the next rising edge of ACLK.

Data Interface

The core receives and transmits data using AXI4-Stream interfaces that implement a video protocol as defined in UG761, *Xilinx AXI Reference Guide*, Video IP: "AXI Feature Adoption."

AXI4-Stream Signal Names and Descriptions

Table 2-2 describes the AXI4-Stream signal names and descriptions

Table 2-8: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
s_axis_currframe_tdata	In	16	Current frame input video data
s_axis_currframe_tvalid	In	1	Current frame input video valid signal
s_axis_currframe_tready	Out	1	Current frame input ready
s_axis_currframe_tuser	In	1	Current frame input video Start-of-Frame signal
s_axis_currframe_tlast	In	1	Current frame input video End-of-Line signal
s_axis_prevframe_tdata	In	16	Previous frame input video data

Table 2-8: **AXI4-Stream Data Interface Signal Descriptions *(Cont'd)***

Signal Name	Direction	Width	Description
s_axis_prevframe_tvalid	In	1	Previous frame input video valid signal
s_axis_prevframe_tready	Out	1	Previous frame input ready
s_axis_prevframe_tuser	In	1	Previous frame input video Start-of-Frame signal
s_axis_prevframe_tlast	In	1	Previous frame input video End-of-Line signal
m_axis_output_tdata	Out	16	Downstream video output data
m_axis_output_tvalid	Out	1	Downstream video output valid signal
m_axis_output_tready	In	1	Downstream video output ready
m_axis_output_tuser	Out	1	Downstream video output Start-of-Frame signal
m_axis_output_tlast	Out	1	Downstream video output End-of-Line signal
m_axis_mem_tdata	Out	16	Memory (temporal feedback) video output data
m_axis_mem_tvalid	Out	1	Memory (temporal feedback) video output valid signal
m_axis_mem_tready	In	1	Memory (temporal feedback) video output ready
m_axis_mem_tuser	Out	1	Memory (temporal feedback) video output Start-of-Frame signal
m_axis_mem_tlast	Out	1	Memory (temporal feedback) video output End-of-Line signal
m_axis_motion_tdata	Out	16	Motion output data
m_axis_motion_tvalid	Out	1	Motion output valid signal
m_axis_motion_tready	In	1	Motion output ready
m_axis_motion_tuser	Out	1	Motion output Start-of-Frame signal
m_axis_motion_tlast	Out	1	Motion output End-of-Line signal

Video Data

The MANR core may handle 8-bit YC data only. The corresponding AXI4-Stream TDATA width is fixed at 16 bits to accommodate 8 Luma and 8 bits Chroma.

READY/VALID Handshake

A valid transfer occurs whenever READY, VALID, ACLKEN, and ARESETn are high at the rising edge of ACLK, as seen in Figure 2-2. During valid transfers, DATA only carries active video data. Blank periods and ancillary data packets are not transferred via the AXI4-Stream video protocol.

Guidelines on Driving tvalid into Slave (Data Input) Interfaces

Once tvalid is asserted, no interface signals except the core's driving tready output may change value until the transaction completes (tready, tvalid high on the rising edge of ACLK). Once asserted, tvalid may only be de-asserted after a transaction has completed. Transactions may not be retracted or aborted. In any cycle following a transaction, tvalid can either be de-asserted or remain asserted to initiate a new transfer.

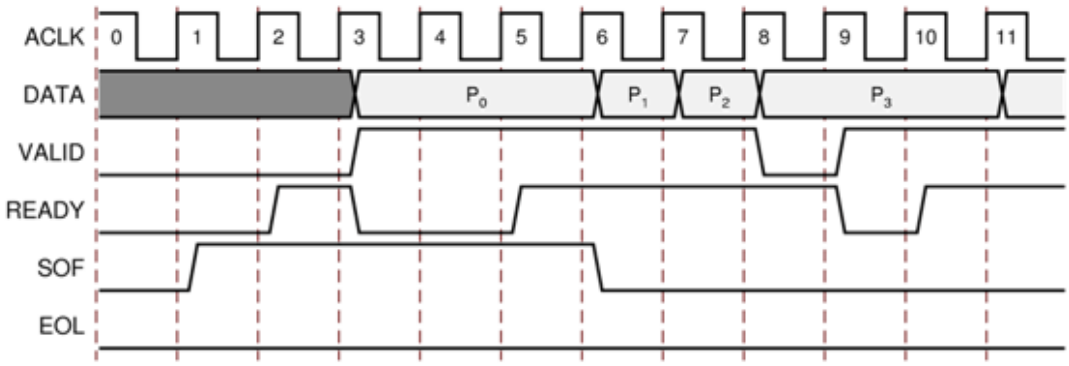

Figure 2-2: **Example of READY/VALID Handshake, Start of a New Frame**

Guidelines on Driving tready into Master (Data Output) Interfaces.

The TREADY signal may be asserted before, during or after the cycle in which the core asserted TVALID. The assertion of TREADY may be dependent on the value of TVALID. A slave that can immediately accept data qualified by this TVALID signal should pre-assert its slave TREADY signal until data is received.

Alternatively, TREADY can be registered and driven the cycle following TVALID assertion. It is recommended that the AXI4-Stream slave should drive TREADY independently, or pre-assert TREADY to minimize latency.

Start of Frame Signals: m_axis_video_tuser0, s_axis_video_tuser0

The Start-Of-Frame (SOF) signal, physically transmitted over the AXI4-Stream TUSER0 signal, marks the first pixel of a video frame. The SOF pulse is one valid transaction wide, and must coincide with the first pixel of the frame, as seen in Figure 2-2. SOF serves as a frame synchronization signal, which allows downstream cores to re-initialize, and detect the first pixel of a frame. The SOF signal can be asserted an arbitrary number of ACLK cycles before the first pixel value is presented on TDATA, as long as a TVALID is not asserted.

End of Line Signals: m_axis_video_tlast, s_axis_video_tlast

The End-Of-Line signal, physically transmitted over the AXI4-Stream TLAST signal, marks the last pixel of a line. The EOL pulse is one valid transaction wide, and must coincide with the last pixel of a scan-line, as seen in Figure 2-3.

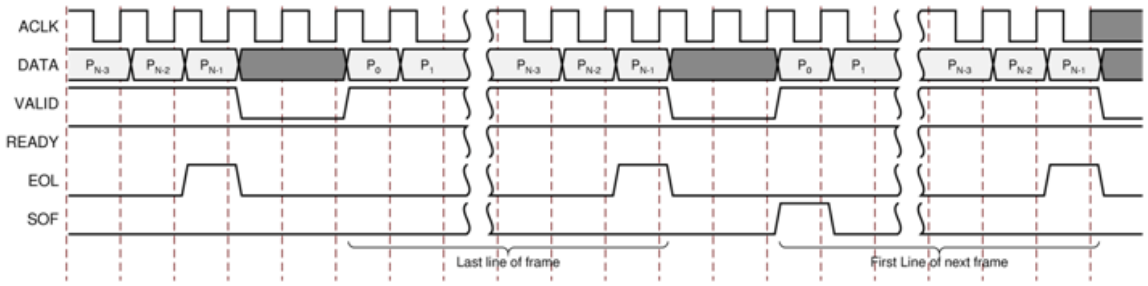

Figure 2-3: **Use of EOL and SOF Signals**

Control Interface

When configuring the core, there is an option to add an AXI4-Lite register interface to dynamically control the behavior of the core. The AXI4-Lite slave interface facilitates integrating the core into the processor system along with other AXI4-Lite compliant IP.

In a static configuration with a fixed set of parameters (constant (fixed-mode) configuration), the core can be instantiated without the AXI4-Lite control interface, which reduces the core footprint.

Constant Configuration

The constant configuration caters to users who will interface the core to a fixed input video source. In constant configuration, parameters such as the image resolution (number of active pixels per scan line and the number of active scan lines per frame) and the Noise-Reduction strength are hard coded into the core through the CORE Generator GUI. Because there is no AXI4-Lite interface, the core is not programmable, but can be reset using the ARESETn port.

AXI4-Lite Interface

The AXI4-Lite interface enables dynamic control of parameters within the core. Core configuration can be accomplished using an AXI4-Stream master state machine, or an embedded ARM or soft system processor such as a MicroBlaze processor. The core can be controlled through the AXI4-Lite interface using read and write transactions to the register space. The AXI4-Lite interface signals are listed in Table 2-9.

Table 2-9: **AXI4-Lite Control Bus Signals**

Name	Direction	Description
S_AXI_AWADDR	In	AXI4-Lite Write Address Bus. The write address bus gives the address of the write transaction.
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 2-9: **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.

Register Space

The standardized Xilinx Video IP register space is partitioned into control-, timing-, and core-specific registers, as described in Table 2-10.

Table 2-10: **MANR Registers**

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register description
0x0000	Control	R/W	No	Pwr-on-Rst: 0x0	b0: SW_ENABLE b1: REG_UPDATE b2: MTF_BYPASS b3: Reserved b4: Reserved b5: Reserved B6: Reserved b7: Reserved b31: SW_RESET (1: reset)
0x0004	Status	R/W	No	0	b0: MTF_LOAD_DONE
0x0008	(Reserved)	-	-	-	-

Table 2-10: **MANR Registers *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register description
0x000C	(Reserved)	-	-	-	-
0x0010	Version	R	No	0x0400a000	7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR
0x0014	(Reserved)	-	-	-	-
0x0018	(Reserved)	-	-	-	-
0x001C	(Reserved)	-	-	-	-
0x0020	(Reserved)	-	-	-	-
0x0100	FRAME_SIZE	R/W	Yes	Specified in GUI	12-0: Number of active pixels per line 28-16: Number of active lines per frame
0x0104	MTF_DIn	W	Yes (internal multiple -MTF memory)	0	7-0: MTF Input Data.
0x0108	MTF_Active	R/W	Yes	Specified in GUI	2-0: Internal MTF Bank currently in use by MANR core
0x010C	MTF_Write	R/W	Yes	0	2-0: Internal MTF Bank to which MTF_Din data will be written.

Control (0x0000) register

- Bit 0 of the CONTROL register, SW_ENABLE, facilitates enabling and disabling the core from software. Writing '0' to this bit effectively disables the core halting further operations, which blocks the propagation of all video signals.

  For the AXI4-Lite interface, after Power up, or Global Reset, the SW_ENABLE defaults to 0. The SW_ENABLE flag is not synchronized with the AXI4-Stream interfaces: Enabling or Disabling the core takes effect immediately, irrespective of the core processing status.

- Bit 1 of the CONTROL register, REG_UPDATE is a write-done semaphore for the host processor, which facilitates committing all user and timing register updates simultaneously. Some core registers are double-buffered. For these cases, one set of registers (the processor registers) is directly accessed by the processor interface, while the other set (the active set) is actively used by the core. New values written to the processor registers will get copied over to the active set at the end of the AXI4-Stream frame, if and only if REG_UPDATE is set. Setting REG_UPDATE to 0 before updating multiple register values, then setting REG_UPDATE to 1 when updates are completed

ensures all registers are updated simultaneously at the frame boundary without causing image tearing.

- Bit 2 of the CONTROL register is the MTF_BYPASS control bit. When this bit is set, noise-reduction is switched off.
- Bit 31 of the CONTROL register, SW_RESET facilitates software reset. Setting SW_RESET reinitializes the core to GUI default values, all internal registers and outputs are cleared and held at initial values until SW_RESET is set to 0. The SW_RESET flag is not synchronized with the AXI4-Stream interfaces. Resetting the core while frame processing is in progress is likely to cause image tearing.

STATUS (0x0004) Register

All bits of the STATUS register can be used to request an interrupt from the host processor. In order to facilitate identification of the interrupt source, bits of the STATUS register remain set after an event associated with the particular STATUS register bit, even if the event condition is not present at the time the interrupt is serviced. Bits of the STATUS register can be cleared individually by writing '1' to the bit position to be cleared.

- Bit 0 of the STATUS register, MTF_LOAD_DONE, should be used when loading MTF values into the core. Its use is described in MTF_DIn (0x0104) Register, MTF_Active (0x0108) Register and MTF_Write (0x010C) Register .

IRQ_ENABLE (0x000C) Register

Any bits of the STATUS register can generate a host-processor interrupt request via the IRQ pin. The Interrupt Enable register facilitates selecting which bits of STATUS register will assert IRQ. Bits of the STATUS registers are masked by (AND) corresponding bits of the IRQ_ENABLE register and the resulting terms are combined (OR) together to generate IRQ.

Version (0x0010) Register

Bit fields of the Version Register facilitate software identification of the exact version of the hardware peripheral incorporated into a system. The core driver can take advantage of this Read-Only value to verify that the software is matched to the correct version of the hardware.

FRAME_SIZE (0x0100) Register

The FRAME_SIZE register encodes the number of active pixels per line and the number of active lines per frame. The lower half-word (bits 12:0) encodes the number of active pixels per line. The upper half-word (bits 28:16) encodes the number of active lines per frame.

Supported values for both are between 32 and the values provided by the user in the GUI. In order to avoid processing errors, the user should restrict values written to FRAME_SIZE the range supported by the core instance.

MTF_DIn (0x0104) Register, MTF_Active (0x0108) Register and MTF_Write (0x010C) Register

Here are some Motion Transfer Function (MTF) loading guidelines:

- The user loads the desired motion transfer function into the Motion Transfer LUT through the MTF_DIn port. Loading the MTF involves writing 128 8-bit unsigned values within the range 0 through 255.
- Of the 128 MTF values to be loaded, the first 64 values must be the Luma MTF values.
- The MSB is bit 7.
- There are eight banks available for the MTF.
- It is not necessary to have MTFs loaded into all banks. However, it is important to make sure that the correct bank is selected active use and for writing.
- Active bank selection is handled by setting the MTF_Active register accordingly. Switching will occur after completion of an output frame.
- MTF bank writing and selection is handled by setting the MTF_Write register accordingly.
- Writing any value to this register resets the internal MTF loading process.
- The 128 values must be loaded sequentially, starting at element 0.
- Following a successful transfer of 128 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 re-writing the MTF_Write register as described above.

Interrupt Subsystem

STATUS register bits can trigger interrupts so embedded application developers can quickly identify faulty interfaces or incorrectly parameterized cores in a video system.

When the core is instantiated with an AXI4-Lite Control interface, the optional interrupt request pin (IRQ) is present. Events associated with bits of the STATUS register can generate a (level triggered) interrupt, if the corresponding bits of the interrupt enable register (IRQ_ENABLE) are set. Once set by the corresponding event, bits of the STATUS register stay set until the user application clears them by writing '1' to the desired bit positions. Using this mechanism the system processor can identify and clear the interrupt source.

Without the AXI4-Lite interface the user can still benefit from the core signaling error and status events. By selecting the **Enable INTC Port** check-box on the GUI, the core generates the optional INTC_IF port. This vector of signals gives parallel access to the individual interrupt sources, as shown in Table 2-11. Unlike STATUS and ERROR flags, INTC_IF signals are not held. They stay asserted only while the corresponding event persists.

Table 2-11: **INTC_IF Signal Functions**

INTC_IF Signal	Name	Function
0	curr_eol_error	Indicates that s_axis_currframe_tlast was asserted when the core did not expect it. The expected position of this pulse is defined according to the source video resolution settings.
1	Not Used	
2	curr_sof_error	Indicates that s_axis_currframe_tuser was asserted when the core did not expect it. The expected position of this pulse is defined according to the source video resolution settings.
3	Not Used	
4	prev_eol_error	Indicates that s_axis_prevframe_tlast was asserted when the core did not expect it. The expected position of this pulse is defined according to the source video resolution settings.
5	Not Used	
6	prev_sof_error	Indicates that s_axis_prevframe_tuser was asserted when the core did not expect it. The expected position of this pulse is defined according to the source video resolution settings.
7	Not Used	

In a system integration tool, such as EDK, the interrupt controller INTC IP can be used to register the selected INTC_IF signals as edge triggered interrupt sources. The INTC IP provides functionality to mask (enable or disable), as well as identify individual interrupt sources from software. Alternatively, for an external processor or MCU, the user can custom build a priority interrupt controller to aggregate interrupt requests and identify interrupt sources.

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 or EDK Graphical User Interface (GUI). This section provides a quick reference to parameters that can be configured at generation time. Figure 3-1 shows the CORE Generator GUI main screen. The EDK GUI is shown in Figure 3-2.

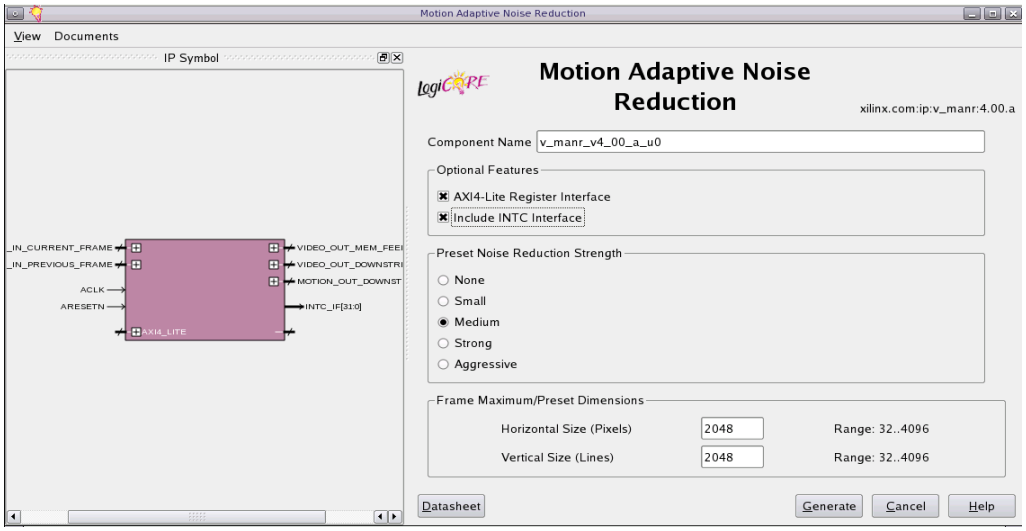

Figure 3-1: **CORE Generator MANR Main Screen**

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_v4_0" is not allowed.

- **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.
  - None: Specifies all zeroes for MTF; bypasses noise reduction such that the output pixel always equals the input pixel.
  - 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
- **Frame Maximum/Preset Dimensions:** These fields represent the maximum anticipated rectangle size on the input and output of the MANR core. The rectangle can vary from 32x32 through 4096x4096. When the core is being used in Fixed mode (AXI4_LITE is disabled), these figures represent the fixed frame dimensions.
- Optional Features:
  - **AXI4-Lite Register Interface:** When selected, the core is generated with an AXI4-Lite interface, which gives access to dynamically program and change processing parameters. For more information, see Control Interface in Chapter 2.
  - **INT_IF interface:** When selected, interrupts are generated on this bus. See Interrupt Subsystem in Chapter 2 for more details.

EDK GUI

Definitions of the EDK GUI controls, shown in Figure 3-2, are identical to the corresponding CORE Generator GUI functions.

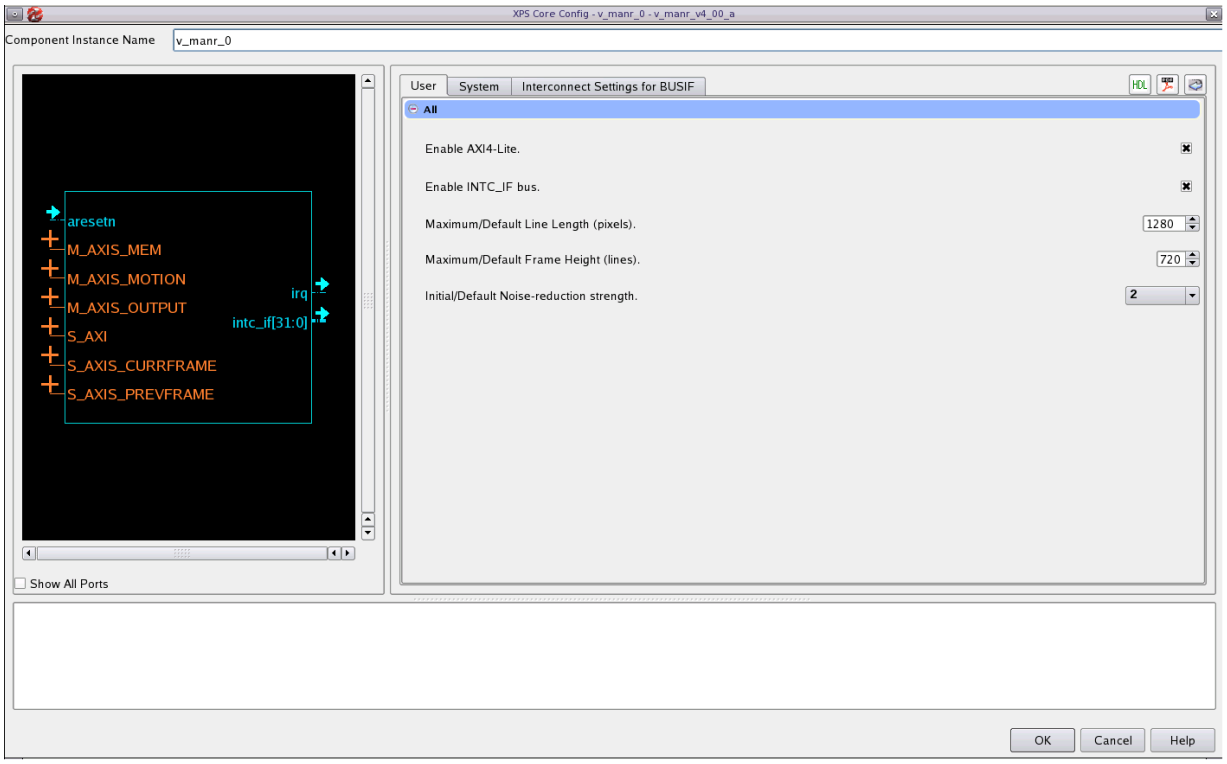

Figure 3-2: **MANR EDK Main Screen**

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 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.

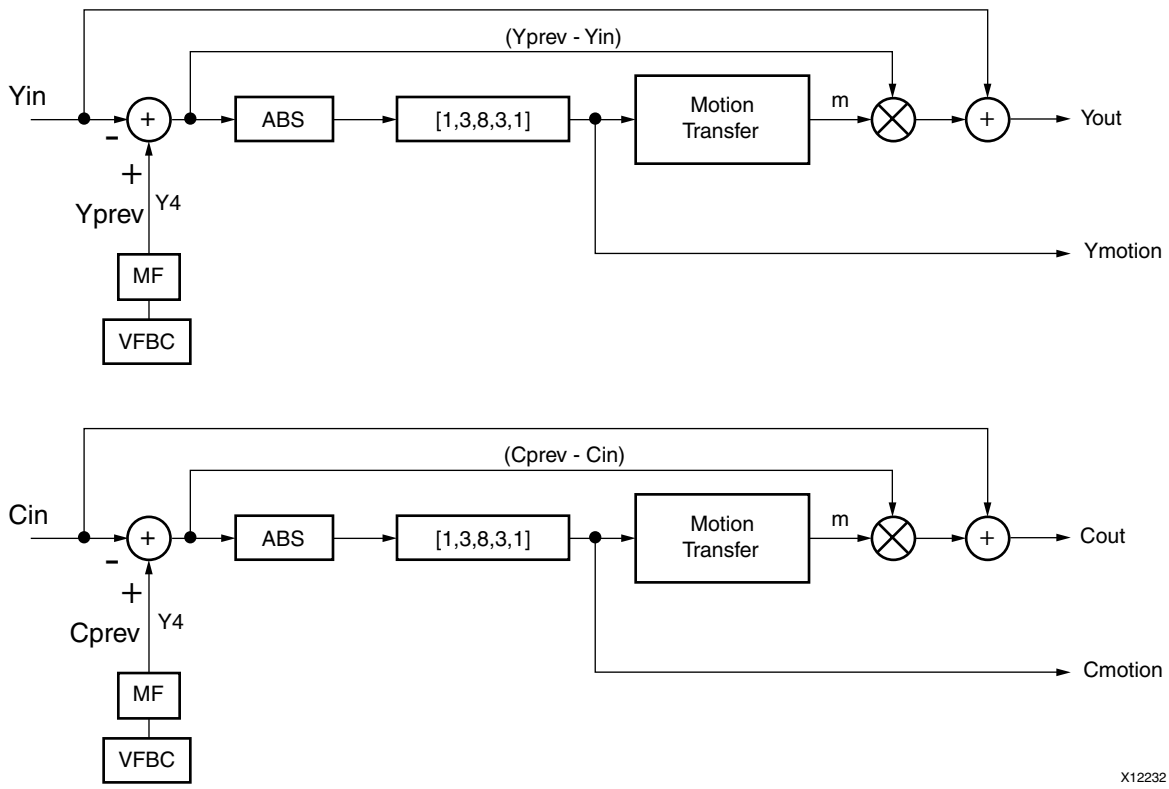

Figure 4-1: **Motion Adaptive Noise Reduction**

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.

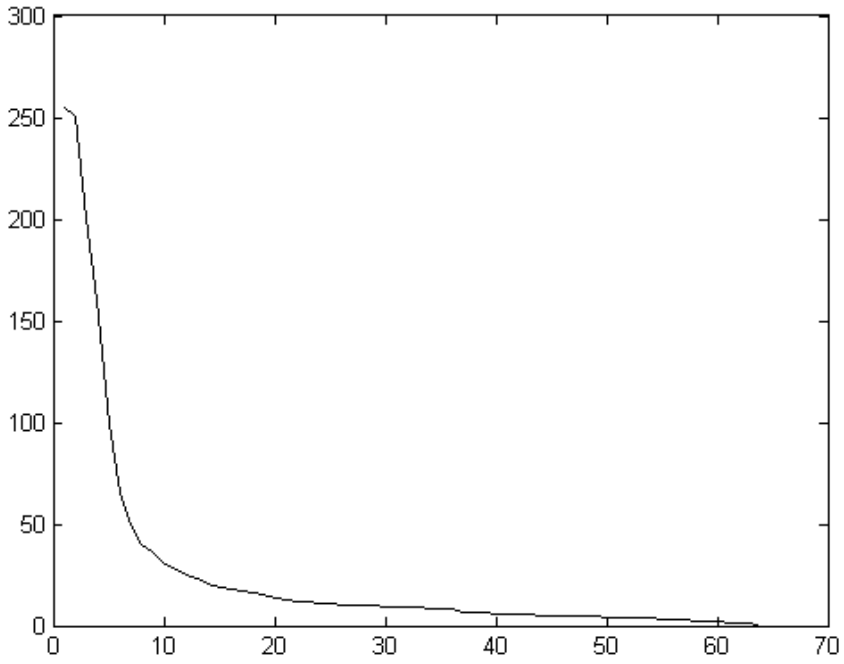

Figure 4-2: **Exponential MTF**

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

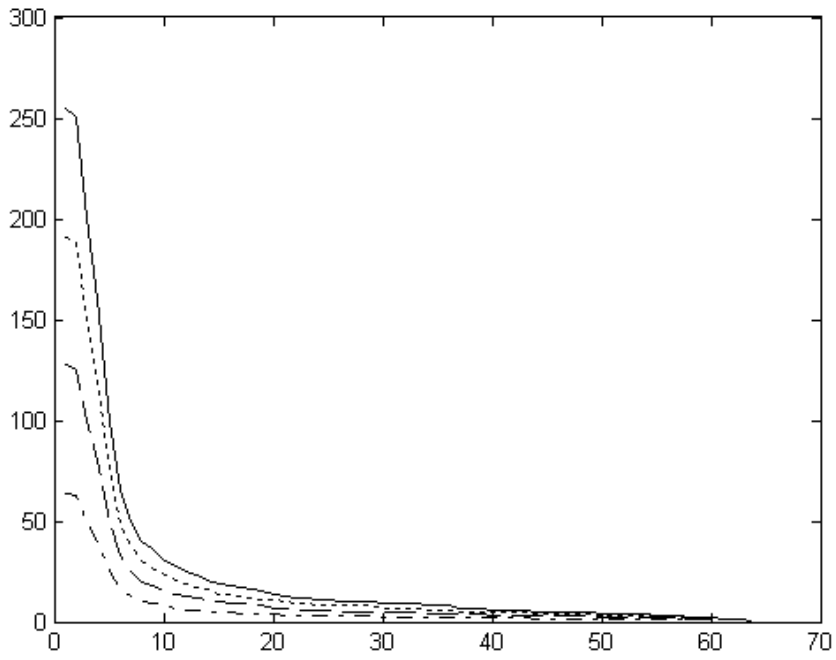

Figure 4-3: **MTF Settings**

The MANR core can store up to eight 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 block RAM internal to the MANR core. The block RAM can store up to eight separate MTF curves. Separate MTF curves are stored for Y and C channels.

Storing 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 another can balance noise reduction and motion artifact from recursion. 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. Using the AXI4-Lite interface enables switching between these curves dynamically.

Xilinx provides five pre-loaded MTF curves by default in the core. In addition, 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 MANR core is the ability to define and use custom MTF curves. To do this successfully, a few properties of MTFs should be well understood. The user can choose to load custom MTF curves into the remaining spaces in the block RAM, or to overwrite the existing ones. By overwriting a default MTF, the default MTF will not be available again unless the FPGA is reprogrammed.

The MTF for each Y and C components 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 Luma (Y) MTF data would appear as shown in Table 4-1, 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.

Table 4-1: **MTF Address Values**

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
32	9	40	6	48	5	56	3
33	8	41	6	49	4	57	2

Table 4-1: **MTF Address Values *(Cont'd)***

Address	Value	Address	Value	Address	Value	Address	Value
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 core through two AXI4-Stream Video protocol 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 core through three AXI4-Stream Video protocol 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 optional 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.

A third AXI4-Stream output interface provides the motion data for optional use by downstream processing blocks. It is available on the m_axis_motion AXI4-Stream interface. Exactly the same as video data, the interface handles 8-bit YC data, transmitted as the lowest 16-bits of the tData element of the output AXI4-Stream bus. Luma motion occupies bits 7:0, and chroma motion occupies bits 15:8.

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

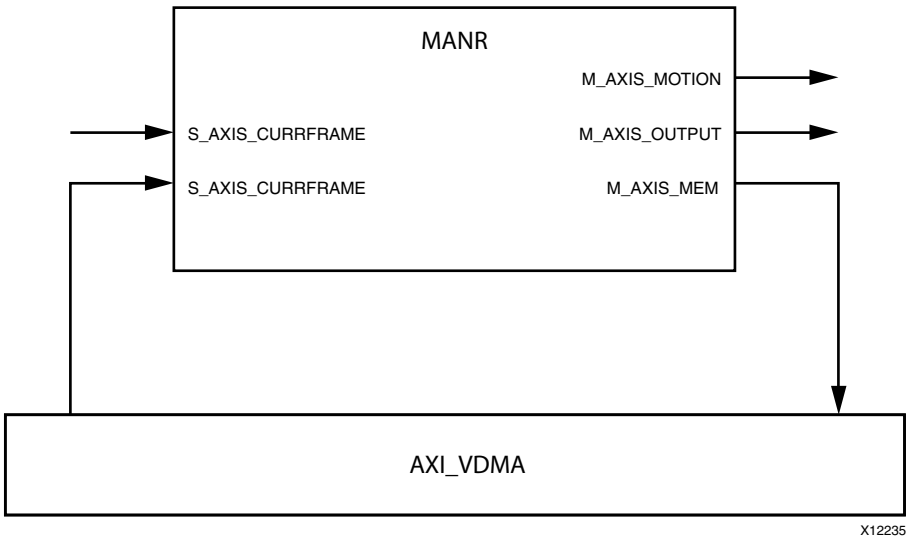

Figure 4-4: **Typical MANR Connectivity**

Interrupts

The MANR core 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 cores. 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 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.

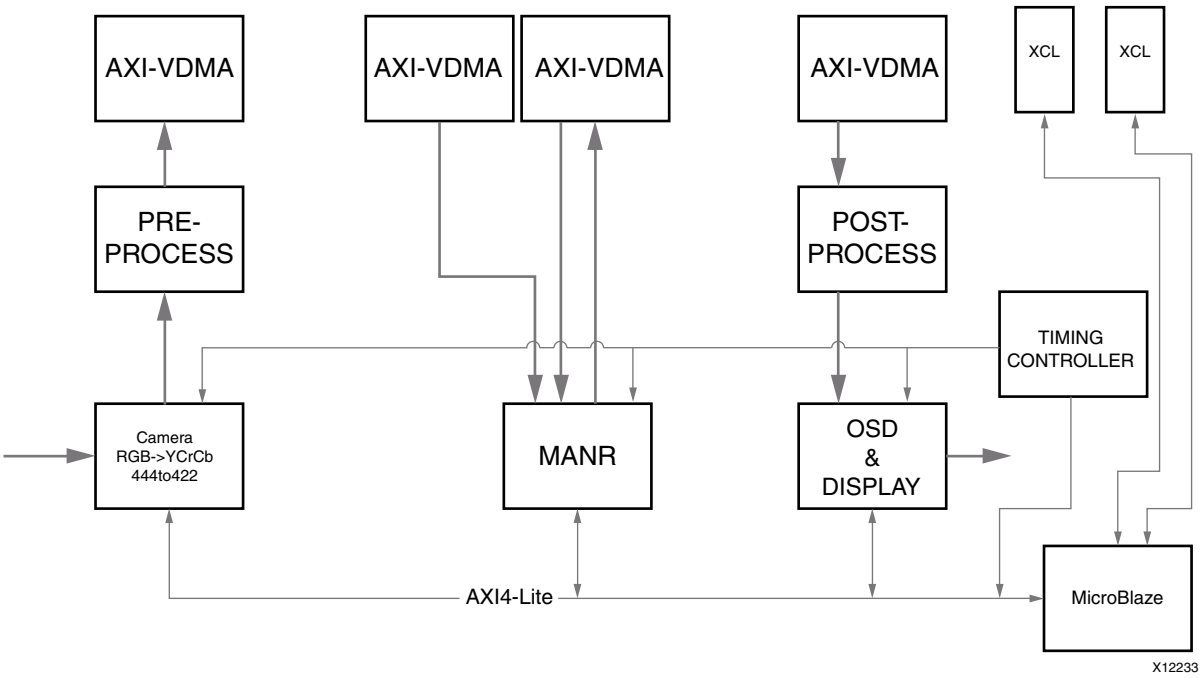

Figure 4-5: **Simple Noise Reduction**

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 ports m_axis_output and m_axis_motion 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.

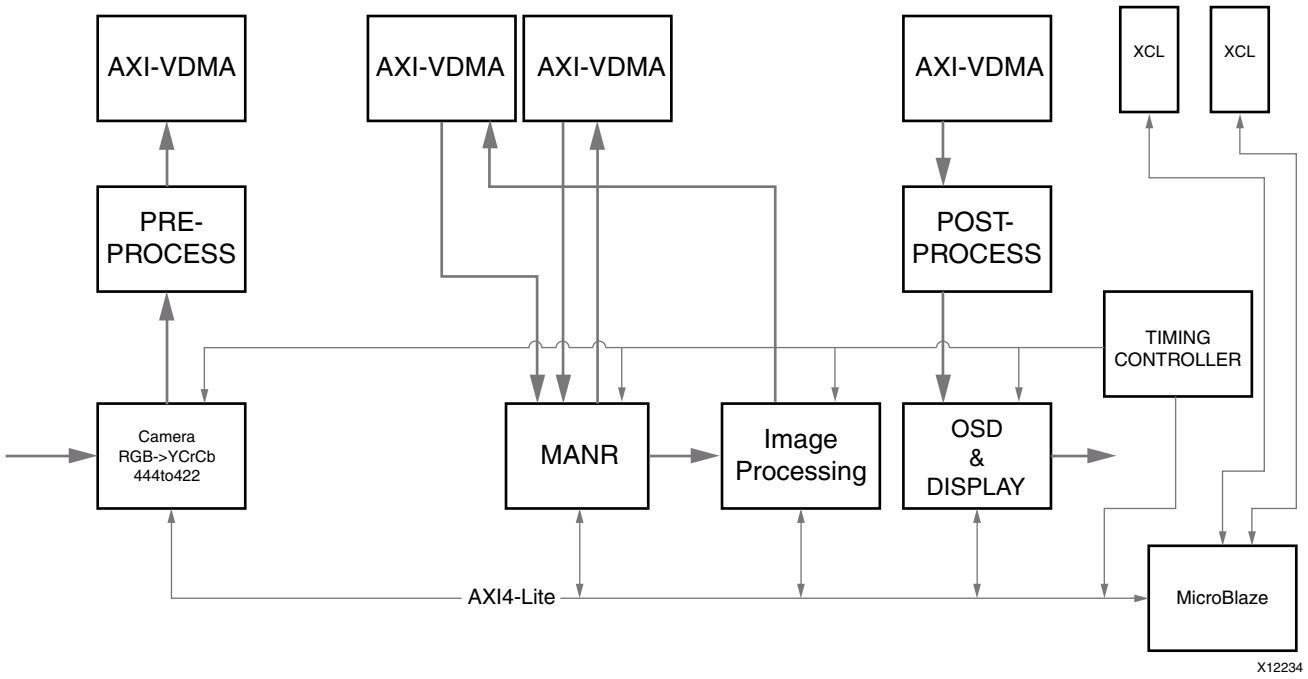

Figure 4-6: **Noise Reduction and Motion Processing**

Clocking

The MANR core has one clock (`aclk`) 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, when using the AX4-Lite Control interface, 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.

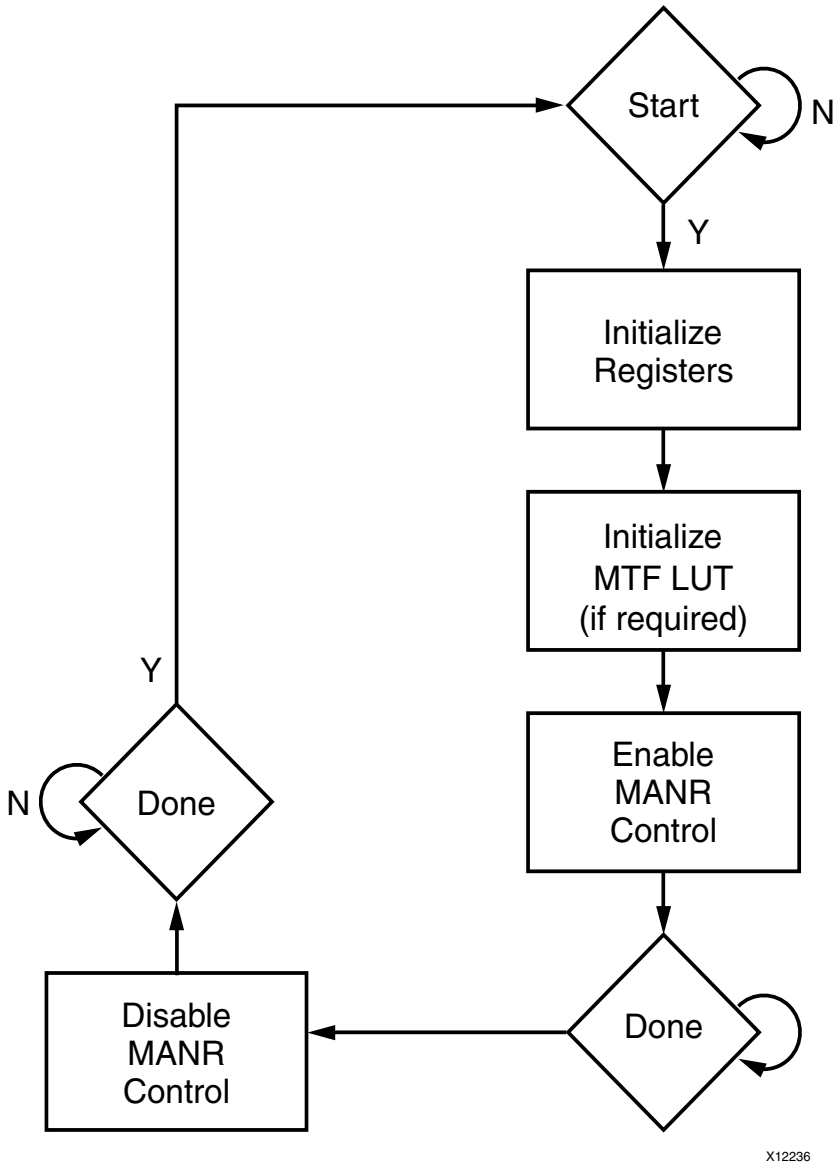

Figure 4-7: **MANR Initialization**

Resets

The MANR core has one reset (`sclr`) that is used for the entire core. When using AXI4-Lite, an internal software reset drives this signal (active Low).

Protocol Description

The register interface is compliant with the AXI4-Lite interface. The video and motion input and output interfaces are compliant with AXI4-Stream Video protocol.

Constraining the Core

This chapter contains applicable constraints for the MANR core.

Required Constraints

The ACLK pin should be constrained at the pixel clock rate desired for the video stream.

Device, Package, and Speed Grade Selections

There are no device, package or speed grade requirements for this core. For a complete list of supported devices, see the release notes for this core.

Clock Frequencies

The pixel clock frequency is the required frequency for this core. See Maximum Frequency in Chapter 2.

Clock Management

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

Clock Placement

There are no specific Clock placement requirements for this core.

Banking

There are no specific Banking requirements for this core.

Transceiver Placement

There are no transceiver placement requirements for this core.

I/O Standard and Placement

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

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
- 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 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.

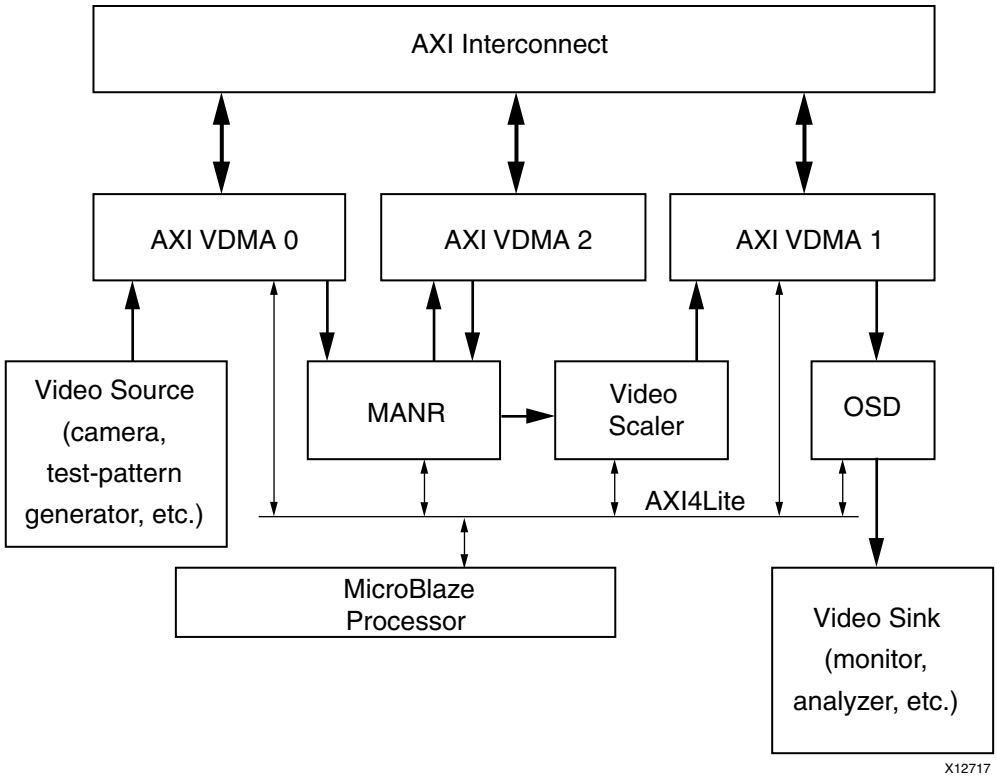

Figure 6-1: **Simplified System Diagram**

Control Buses

In this example, the MicroBlaze processor is configured to use the AXI4-Lite bus. The AXI-VDMAs, MANR, Video Scaler, 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.

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.

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

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

Demonstration Test Bench

A demonstration test bench is provided which enables core users to observe core behavior in a typical use scenario. The user is encouraged to make simple modifications to the test conditions and observe the changes in the waveform.

Test Bench Structure

The top-level entity, `tb_main.v`, instantiates the following modules:

- DUT

  The core instance under test.

- axi4lite_mst

  The AXI4-Lite master module, which initiates AXI4-Lite transactions to program core registers.

- axi4s_video_mst

  The AXI4-Stream master module, which opens the stimuli TXT file and initiates AXI4-Stream transactions to provide stimuli data for the core.

- axi4s_video_slv

  The AXI4-Stream slave module, which opens the result TXT file and verifies AXI4-Stream transactions from the core.

Running the Simulation

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

Directory and File Contents

The directory structure underneath the top-level folder is:

- expected:
  Contains the pre-generated expected/golden data used by the test bench to compare actual output data.
- stimuli:
  Contains the pre-generated input data used by the test bench to stimulate the core (including register programming values).
- results:
  Actual output data will be written to a file in this folder.
- src:
  Contains the VHD simulation files and the .xco CORE Generator parameterization file of the core instance. The VHD file is a netlist generated using CORE Generator. The XCO file can be used to regenerate a new netlist using CORE Generator.

The available core C-model can be used to generate stimuli and expected results for any user BMP image. For more information, refer to Appendix E, C Model Reference.

The top-level directory contains packages and Verilog modules used by the test bench, as well as:

- isim_wave.wcfg:
  Waveform configuration for ISIM
- mti_wave.do:
  Waveform configuration for ModelSim
- run_isim.bat :
  Runscript for iSim in Windows
- run_isim.sh:
  Runscript for iSim in Linux
- run_mti.bat:
  Runscript for ModelSim in Windows
- run_mti.sh:
  Runscript for ModelSim in Linux

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.

Appendix B

Migrating

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

From v3.0 to v4.00.a of the MANR core, the following significant changes took place:

- AXI4-Stream Interface usage was updated. It no longer supports the AXI4-Stream tKeep usage. It now uses the AXI4-Stream tUser pin as a Start-of-Frame indicator.
- The GPP control option has been removed. The core may now only be controlled dynamically by using the AXI4-Lite interface.
- The core now supports a Constant (Fixed-mode) option whereby parameters such as image size and noise-reduction level are specified in the configuration GUI. The user may select from one of the pre-loaded MTF curves ("None", "Weak", "Medium", "Strong" or "Aggressive").
- Core Feature Changes:
  - The MANR v3.0 provided the motion data output as part of the m_axis_output bus. In v4.00.a, the core now supports an independent AXI4-Stream output for the motion data. The v4.00.a core uses two input AXI4-Stream interfaces (current and previous frames) plus three output AXI4-Stream interfaces (downstream video, temporal feedback video, and downstream motion).
  - In v3.0, the MTF table was common for Y and C processing. The core now supports independent Y and C MTF curve storage.
  - As with v3.0, the v4.00.a core includes five pre-loaded MTF curves. However, the v4.00.a core provides dynamic access to all the pre-loaded MTF curves using the AXI4-Lite control interface.

Debugging

Some general debugging tips are as follows:

- Verify the Version register can be read properly. See Table 2-10, page 14 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 start-of-frame pulse on the s_axis_xxx_tuser input is being properly driven for both the currframe and prevframe AXI4-Stream interfaces.
- 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.

It is recommended to prototype the system with the AXI4-Stream interface enabled, so that status/error detection, reset, and dynamic size programming can be used during debugging.

The following steps are recommended to bring up/debug the core in a video/imaging system:

1. Bring up the AXI4-Lite interface.
2. Bring up the AXI4-Stream interfaces.
3. Find the right noise-reduction value.

Once the core is working as expected, the user may consider "hardening" the configuration by replacing the core with an instance where GUI default values are set to the established FRAME_SIZE and NOISE_REDUCTION values, but the AXI4-Lite interface is disabled. This configuration reduces the core slice footprint, and reduces SW burden. It could potentially lead to the removal of any software element if a fixed-mode system is ultimately desired.

Bringing up the AXI4-Lite Interface

Table C-1 describes how to troubleshoot the AXI4-Lite interface.

Table C-1: **Troubleshooting the AXI4-Lite Interface**

Symptom	Solution
Readback value for the VERSION_REGISTER is different from expected default values.	Does the core receive ACLK? Is the core in reset? Set ARESETn=1. The address maps between software and hardware could get out of sync. Regenerate addresses in EDK, make sure the MHS file in EDK and the xparameters.h in the SDK project are up to date.
Readback values from registers are stuck, and cannot be overwritten.	Is the target address writable? Make sure that bit 1 an Control register (0x0000) is set.
The interface is unreliable. Subsequent reads from the same address return different value; readback values differ from values written to the same address.	Clock domain crossing issues between the host processor and the peripheral. HDL users need to make sure the AXI4-Lite Master is in the same clock domain as the AXI4-Lite Slave. If not, proper clock-domain crossing logic, such as asynchronous FIFOs need to be inserted.

If the AXI4-Stream communication is sound, but the data seems corrupted, the next step is to verify the configuration for the MANR core. Please check that the FRAME_SIZE register has been set correctly.

Bringing up the AXI4-Stream Interfaces

Table C-2 describes how to troubleshoot the AXI4-Lite interface.

Table C-2: **Troubleshooting AXI4-Stream Interface**

Symptom	Solution
Bit 0 of the ERROR register reads back set.	Bit 0 of the ERROR register, EOL_EARLY, indicates the number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Chipscope, measure the number of active AXI4-Stream transactions between EOL pulses.
Bit 1 of the ERROR register reads back set.	Bit 1 of the ERROR register, EOL_LATE, indicates the number of pixels received between the last End-Of-Line (EOL) signal surpassed the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Chipscope, measure the number of active AXI4-Stream transactions between EOL pulses.
Bit 2 or Bit 3 of the ERROR register reads back set.	Bit 2 of the ERROR register, SOF_EARLY, and bit 3 of the ERROR register SOF_LATE indicate the number of pixels received between the latest and the preceding Start-Of-Frame (SOF) differ from the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Chipscope, measure the number EOL pulses between subsequent SOF pulses.
s_axis_video_tready stuck low, the upstream core cannot send data.	During initialization, line-, and frame-flushing, the CFA core keeps its s_axis_video_tready input low. Afterwards, the core should assert s_axis_video_tready automatically. Is m_axis_video_tready low? If so, the CFA core cannot send data downstream, and the internal FIFOs are full.
m_axis_video_tvalid stuck low, the downstream core is not receiving data	1. No data is generated during the first two lines of processing. 2. If the programmed active number of pixels per line is radically smaller than the actual line length, the core drops most of the pixels waiting for the (s_axis_video_tlast) End-of-line signal. Check the ERROR register.
Generated SOF signal (m_axis_video_tuser0) signal misplaced.	Check the ERROR register.
Generated EOL signal (m_axis_video_tlast) signal misplaced.	Check the ERROR register.
Slave (input-side) tready output(s) stuck low; the upstream core cannot send data.	During initialization, the core drives its slave tready signals low. Afterwards, the core should assert tready automatically. Are all Master tready signals being driven into the core High? If not, the MANR core cannot send data downstream, and the internal FIFOs are full.

Table C-2: **Troubleshooting AXI4-Stream Interface**

Symptom	Solution
Master (output) tvalid signal(s) stuck low; the downstream core is not receiving data.	Are the upstream tready signals high on all 3 Master (output) AXI4-Stream interfaces? Is the MANR core actually receiving data at it's input interfaces?
Data samples lost between Upstream core and the MANR core, or between MANR core and downstream core. Inconsistent EOL and or SOF periods received by downstream core.	1. Are the Master and Slave AXi4-Stream interfaces in the same clock domain? If not, is proper clock-domain crossing logic (Asynchronous FIFO) instantiated appropriately? 2. Did the design meet timing? 3. Is the frequency of the clock source driving the MANR ACLK pin lower than the reported Fmax reached?

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.

Application Software Development

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

Device Drivers

EDK pCore Driver Files

The MANR EDK 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.

EDK pCore API Functions

This section describes the functions included for the EDK 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.

C Model Reference

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

- Bit accurate with v_manr_v4_00_a 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 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 53.

The main features of the C model package are:

- **Bit Accurate C Model** - produces the same output data as the MANR 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_v4_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_v4_00_a_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_v4_00_a_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_v4_00_a_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.

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

File Name	Contents
libIp_v_manr_v4_00_a_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_v4_00_a_bitacc_cmodel.so
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_v4_00_a_bitacc_cmodel.dll	Precompiled library file for 64-bit Windows platforms compilation
libIp_v_manr_v4_00_a_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_v4_00_a_bitacc_cmodel.dll	Precompiled library file for 32-bit Windows platforms compilation
libIp_v_manr_v4_00_a_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 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 2008

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_v4_00_a_generics generics,
      struct xilinx_ip_v_manr_v4_00_a_inputs inputs,
      struct xilinx_ip_v_manr_v4_00_a_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_v4_00_a_generics manr_generics;
struct  xilinx_ip_v_manr_v4_00_a_inputs  manr_inputs;
struct xilinx_ip_v_manr_v4_00_a_outputs* manr_outputs
```

The declaration of these structures are in the `v_manr_v4_00_a_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 56 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 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.

Table E-3: **Member Variables of the Video Structure**

Member Variable	Designation
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)
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 58. 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 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_v4_00_a_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_v4_00_a_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_v4_00_a_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_v4_00_a_bitacc_cmodel.h`
- `libIp_v_manr_v4_00_a_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_v4_00_a_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_v4_00_a_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.

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.

For a comprehensive listing of Video and Imaging application notes, white papers, reference designs and related IP cores, see the Video and Imaging Resources page:

www.xilinx.com/esp/video/refdes_listing.htm

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

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release.
04/24/2012	2.0	Updated core to v4.00a and ISE Design Suite to v14.1.

Notice of Disclaimer