LogiCORE IP Motion Adaptive Noise Reduction v2.0

DS841 March 1, 2011 **Product Specification**

Introduction

The Xilinx Motion Adaptive Noise Reduction (MANR) LogiCORE™ IP is a module for both motion detection and motion adaptive noise reduction in video systems. The core allows the motion detection function to be used independently of the noise reduction function for applications where noise reduction is not needed. The noise reduction algorithm is implemented as a recursive temporal filter with a user programmable transfer function allowing the user to control both the shape of the motion transfer and the strength of the noise reduction applied. The motion transfer function is initialized according to the settings in the CORE Generator™ GUI, but is also programmable at runtime via the register interface. CORE Generator™ technology generates the core as either an AXI EDK pCore, a standalone netlist for a General Purpose Processor (GPP) or as a Fixed Mode netlist. When generated as an EDK pCore, the processor interface is AXI4-Lite compliant.

Features

- Programmable register control
- Selectable processor interface
  - EDK pCore - AXI interface is based on AXI4-Lite specification
  - General Purpose Processor
- Selectable and programmable motion transfer function
- Full support for interrupts and status registers for easy system control.
- Supports YUV 4:2:2, 4:2:0 at 8 bits per pixel
- Support for HD frame sizes including 720x480, 1280x720, 1920x1080
- Support for 720p60 and 1080p30

LogiCORE IP Facts Table					
Core Specifics					
Supported Device Family(1)	Spartan®-3A DSP, Spartan-6LX, Spartan-6LXT, Virtex®-5, Virtex-6LX, Virtex-6LXT				
Supported User Interfaces	General Purpose Processor (GPP), EDK pCore AXI4-Lite, Constant				
	Resources				**Frequency**
Configuration	**LUTs**	**FFs**	**DSP Slices**	**Block RAMs(2)**	**Max. Freq.(3)**
Spartan-6	<570	<850	3	1	150 MHz
Virtex-5	<570	<850	3	1	225 MHz
Virtex-6	<570	<850	3	1	225 MHz
Provided with Core					
Documentation	Product Specification				
Design Files	Netlist for GPP and Constant interfaces, Encrypted source code for EDK pCore				
Example Design	Not Provided				
Test Bench	Provided				
Constraints File	Not Provided				
Simulation Model	Bit Accurate C Model				
Tested Design Tools					
Design Entry Tools	CORE Generator™, Platform Studio (XPS)				
Simulation	ModelSim v6.6d, Xilinx ISIM 13.1				
Synthesis Tools	ISE® 13.1				
Support					
Provided by Xilinx, Inc.					

1. For a complete listing of supported devices, see the release notes for this core.
2. Based on 18K block RAMs (or 36K - select appropriate size).
3. For more complete performance data, see Performance, page 25.

Applications

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

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 most likely due to noise in the current frame, and therefore the previous frame can be used. This recursive action effectively reduces noise while preserving the output image content by masking small changes but preserving larger pixel changes.

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

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 in luma for 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. To simplify the filter and reduce resources, the same MTF value calculated for the luma, is sub-sampled and applied to the chroma values. Consequently, the entire input frame is filtered in a recursive fashion as shown in Figure 1.

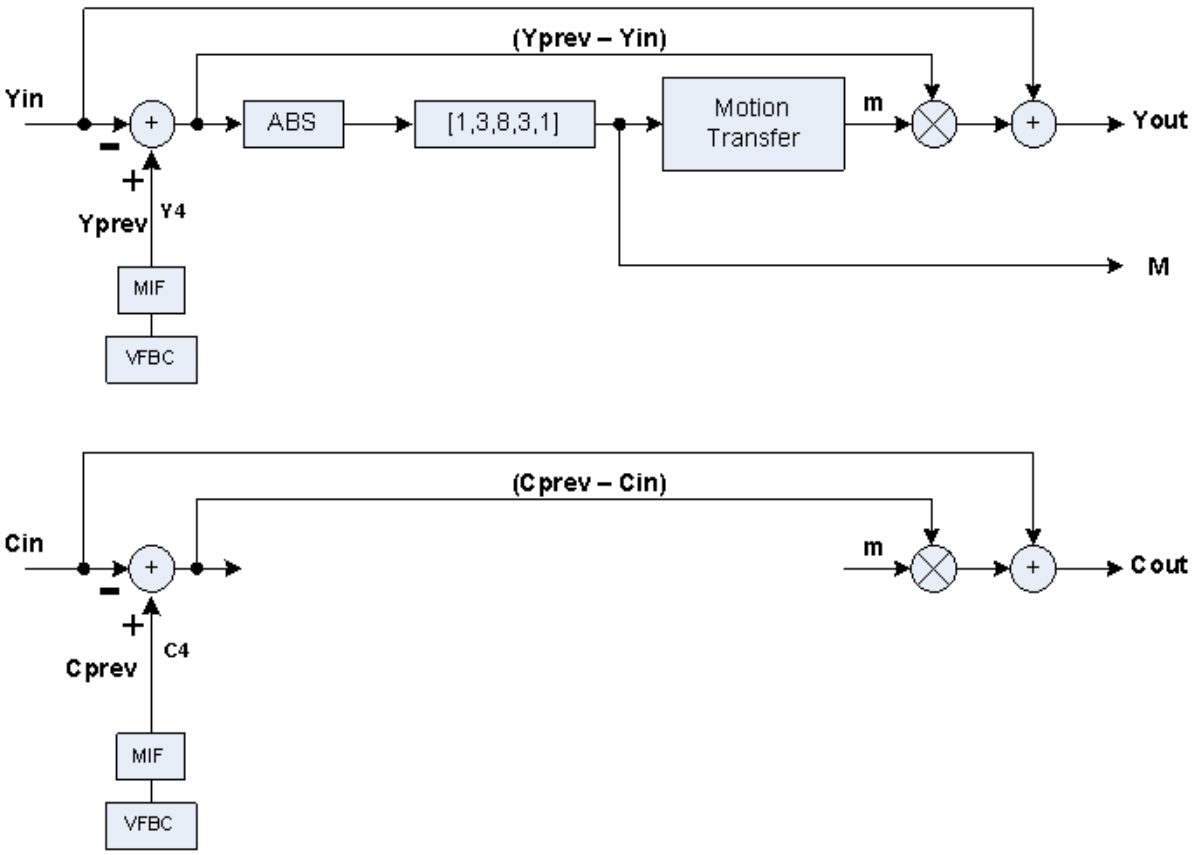

Figure 1: **Motion Adaptive Noise Reduction**

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 2. The Y-axis denotes the amount of recursion, and the X-axis denotes the amount of motion.

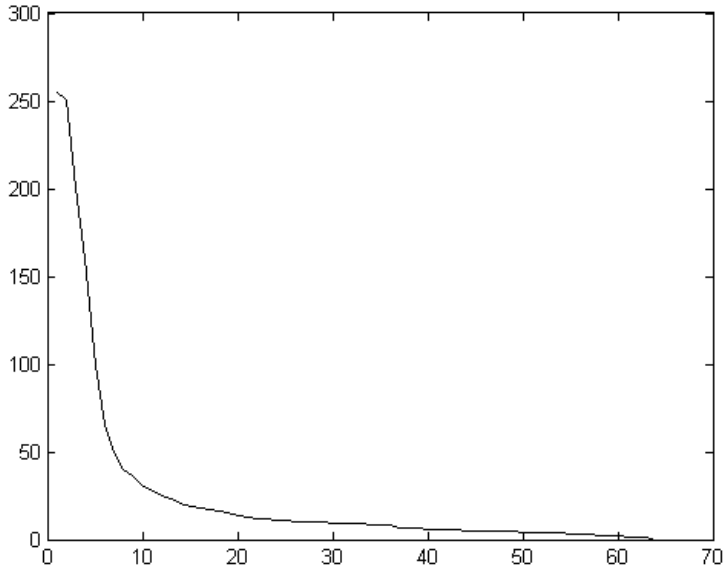

Figure 2: **Exponential MTF**

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 2. This results in the same shape response, but varying degrees of recursion for the same shape. Shown in Figure 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 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 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

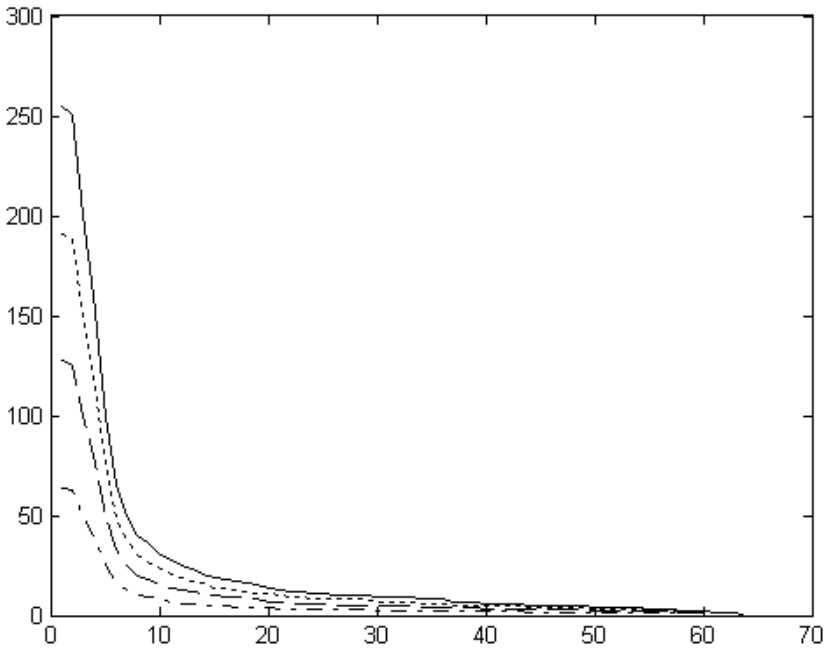

Figure 3: **MTF Settings**

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

Video Frame Buffer Controller (VFBC) Interface Overview

The MANR core 2.0 uses a dedicated VFBC interface. All video data for the MANR is handled via the VFBC interface. The VFBC is a personality interface module for the Xilinx multi-port memory controller (MPMC). These two memory interface cores support the requirement of the MANR to have access to both a current and previous line of video from the same frame, while avoiding extensive consumption of the internal FPGA RAM resources for internal buffering.

Through the single VFBC port, the MANR reads the current and previous video lines and writes the resulting video line. This data does not contain the raw motion scalar values. The motion data can be accessed through the YCM data interface.

YCM Data Interface Overview

In addition to the VFBC interface, the MANR core also provides a simple interface for accessing the motion data. The luma and chroma are also available on this port. The YCM data interface provides, as a single 24-bit word, the luma, chroma, and motion data along with a data qualifier signal, `YCMOut_WE`. This interface can be used to source data to other processing cores that may perform additional calculation such as statistics, object tracking, etc.

MTF Storage and Switching

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

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

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

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

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

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

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

CORE Generator Graphical User Interface (GUI)

The Xilinx Motion Adaptive Noise Reduction (MANR) LogiCORE IP is easily configured to meet the developer's specific needs through the CORE Generator graphical user interface (GUI). This section provides a quick reference to parameters that can be configured at generation time. Figure 4 shows the GUI main screen.

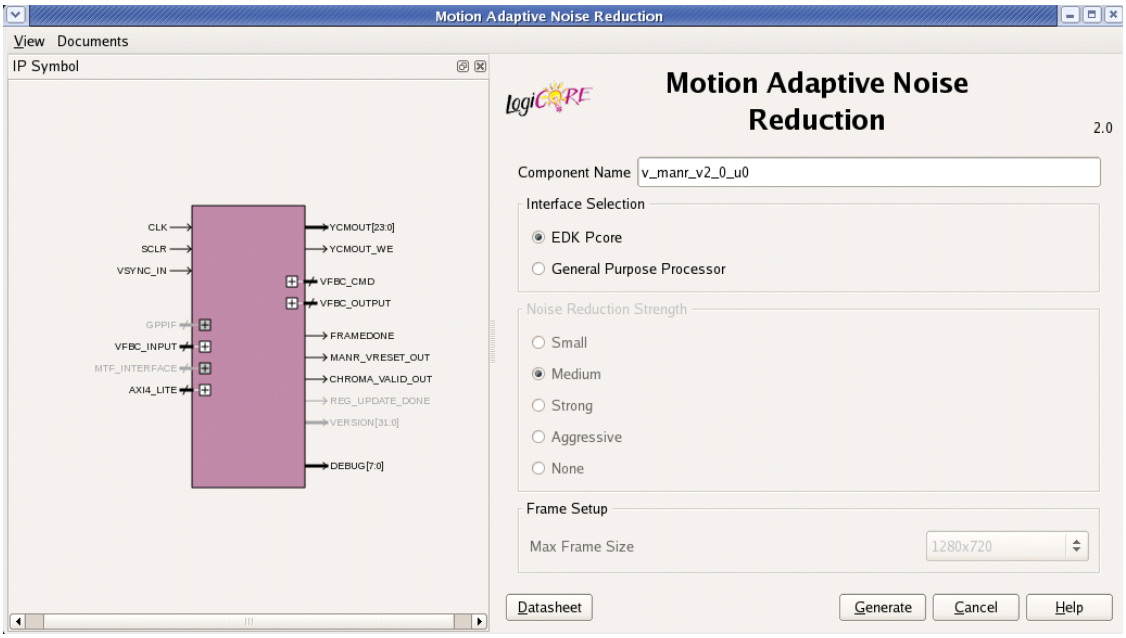

Figure 4: **MANR Main Screen**

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_v1_1" is not allowed.

- Interface Selection: The MANR is generated with one of two interfaces.
  - **EDK pCore Interface:** CORE Generator software generates the MANR as a pCore that can be easily imported into an EDK project as a hardware peripheral. The core registers can then be programmed in real-time via a MicroBlaze processor. See the EDK pCore Interface section for more information.
  - **General Purpose Processor Interface:** CORE Generator software generates a set of ports that can be used to program the MANR. See the General Purpose Processor Interface section for more information.
- **Noise Reduction Strength:** This parameter selects the default MTF. The MTF is initialized according to one of the following settings. The MTF is fully programmable, and the initial values specified during core generation can easily be overridden by programming the desired MTF at run time.
  - Small: Specifies ¼ gain exponential curve for MTF
  - Medium: Specifies ½ gain exponential curve for MTF
  - Strong: Specifies ¾ gain exponential curve for MTF
  - Aggressive: Specifies full gain exponential curve for MTF
  - None: Specifies all zeroes for MTF; bypasses noise reduction such that the output pixel always equals the input pixel.
- **Frame Setup:** This parameter configures the MANR maximum frame dimensions. The following choices are available:
  - 1920x1080
  - 1280x720
  - 720x480
  - 640x480
  - 640x360

MANR Core Interfaces

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

EDK pCore Interface

The pCore interface creates a pCore and device driver that can be easily added to an EDK Project as a hardware peripheral. This section describes the Register Set, the pCore Driver Files, and the I/O signals associated with the MANR pCore.

Migrating to the EDK pCore AXI4-Lite Interface

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

http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

Parameter Modification in CORE Generator Software

EDK pCore parameters found in the axi_manr_v2_00_a/data/manr_v2_1_0.mpd file *cannot* be modified in the Xilinx CORE Generator tool. Parameters shown on the CORE Generator Graphical User Interface will be disabled if the EDK pCore (AXI4-Lite) Interface is selected. Xilinx recommends that all parameter changes be made with the Motion Adaptive Noise Reduction pCore GUI in the EDK environment.

pCore Register Set

The pCore interface provides a memory-mapped interface for the programmable registers within the core, which are defined in Table 1. All registers default to 0x00000000 on Power-on/Reset.

Table 1: **MANR pCore Memory Mapped Register Set**

Address (hex)	Register Name	Access Type	Description	
BASEADDR + 0x0000	MANR Control	R/W	Control	
			31:3	Reserved
			2	Enable bypass mode 1 = Disable noise reduction functionality: Output = input 0 = Enable noise reduction functionality
			1	Enable register updates 1 = MANR core will update registers on next vertical sync
			0	Enable 1 = Enable MANR core on next vertical sync
BASEADDR + 0x0004	MANR Status	R	Status	
			31:2	Reserved
			1	Register update completed 1 indicates that register values for current frame have been accepted by the core. Occurs once per frame. This bit is cleared when any value is written to the register.
			0	MTF load completed 1 indicates that 64 values have been written into the MTF LUT successfully. This bit is cleared when any value is written to the register.
BASEADDR + 0x0008	MANR Error Codes	R	Error	
			24:31	Reserved
			16:23	Reserved
			8:15	Reserved
			0:7	Reserved
BASEADDR + 0x000C	MANR Frame Status	R	Frame processing status	
			31:1	Reserved
			0	Frame done 1 indicates that an entire frame has been processed by the MANR core. This bit is cleared when any value is written to the register.
BASEADDR + 0x0010	VFBC Stride	R/W	Memory space allocated per line of video	
			31:0	Defaults to 2x frame width
BASEADDR + 0x0014	Frame Size	R/W	Active video frame dimensions	
			31:28	Reserved
			27:16	Frame height (defaults to GUI settings)
			15:12	Reserved
			11:00	Frame width (defaults to GUI settings)

Table 1: **MANR pCore Memory Mapped Register Set *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
BASEADDR + 0x0018	MTF Data In	R/W	MTF load data register	
			31:8	Reserved
			0:7	MTF data
BASEADDR + 0x001C	MTF Write Bank	R/W	MTF Write bank control	
			31:1	Reserved
			0	MTF bank to be updated via MTF_data In Default 0
BASEADDR + 0x0024	Previous Frame Base Address	R/W	Previous frame store start location	
			31:0	Memory address of top left hand corner of previous frame Default (0x00000000)
BASEADDR + 0x0028	Output Frame Base Address	R/W	Output frame store start location	
			31:0	Memory address of top left hand corner of output frame Default (0x00000000)
BASEADDR + 0x002C	Current Frame 0 Base Address	R/W	Circular frame buffer location 0, start address	
			31:0	Memory address of top left hand corner of current frame (buffer 0) Default (0x00000000)
BASEADDR + 0x0030	Current Frame 1 Base Address	R/W	Circular frame buffer location 1, start address	
			31:0	Memory address of top left hand corner of current frame (buffer 1) Default (0x00000000)
BASEADDR + 0x0034	Current Frame 2 Base Address	R/W	Circular frame buffer location 2, start address	
			31:0	Memory address of top left hand corner of current frame (buffer 2) Default (0x00000000)
BASEADDR + 0x0038	Current Frame 3 Base Address	R/W	Circular frame buffer location 3, start address	
			31:0	Memory address of top left hand corner of current frame (buffer 3) Default (0x00000000)
BASEADDR + 0x003C	Current Frame 4 Base Address	R/W	Circular frame buffer location 4, start address	
			31:0	Memory address of top left hand corner of current frame (buffer 4) Default (0x00000000)
BASEADDR + 0x00F0	HW Version Register	R		
BASEADDR + 0x0100	Software Reset	R/W	MANR SW reset of core	
			31:1	Reserved
			0	1 resets MANR core
BASEADDR + 0x021C	GIER	R/W	Global interrupt enable register	
			31	Mask to enable global interrupts
			30:0	Reserved

Table 1: **MANR pCore Memory Mapped Register Set *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
BASEADDR + 0x0220	ISR	R	Interrupt status register	
			31:2	Reserved
			1	Edge sensitive interrupt for IP core done with video frame
			0	Reserved
BASEADDR + 0x0228	IER	R/W	Interrupt enable register	
			31:2	Reserved
			1	Mask or enable interrupt for IP core done with video frame
			0	Reserved

pCore Circular Buffer

The MANR pCore includes a 5-frame circular buffer controller to facilitate integration with the Xilinx LogiCORE VFBC. This arrangement supports "genlocking" with the VFBC to manage input and output frames in a complete system. This circular buffer supports "genlocking" with the VFBC to manage input and output frames in a complete system.

This buffer controller consists of five separate address registers (Current Frame 0-4) and a frame buffer pointer. The MANR pCore automatically rotates through these buffers.

pCore Driver Files

The MANR pCore includes software driver files that the user can use to control all of the system registers described in the previous section. Table 2 lists the files included with the MANR pCore.

Table 2: **Software Driver Files Provided with the MANR pCore**

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

pCore Interrupts

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

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

pCore I/O Signals

The I/O signals on the MANR pCore can be broken into three groups: General Interface, VFBC and genlock, and AXI4-Lite signals. See Tables 3, 4, and 5.

Table 3: **pCore General Interface Signals**

Name	Direction	Description
sclr	In	System synchronous reset (active high)
clk	In	Main core clock
YCMOut_WE	Out	Indicates valid data on YCMOut
chroma_valid_out	Out	Indicates validity of 4:2:0 chroma lines on alternate-line basis
YCMOut	Out	Bits[7:0]: Luma out Bits[15:8]: Chroma out Bits[23:16]: Motion out
framedone	Out	Indicates the end of the frame
MANR_vreset_Out	Out	General purpose reset signal asserted for one line period during vertical blanking
Debug	Out	Bit0: Input vsync_in heartbeat; 1 Hz flash for 50 Hz vsync_in Bit1: Input clk heartbeat; 1 Hz flash for 100 MHz clk Bit2: '0' Bit3: '0' Bit4: reset Bit5: VFBC_cmd_almost_full Bit6: VFBC_wd_almost_full Bit7: VFBC_rd_almost_empty

Table 4: **VFBC and Genlock Signals**

Name	Direction	Description
VFBC_rd_clk	Out	VFBC read clock
VFBC_rd_reset	Out	VFBC read data reset
VFBC_rd_read	Out	VFBC read data enable
VFBC_rd_end_burst	Out	VFBC read end burst
VFBC_rd_flush	Out	VFBC read flush
VFBC_rd_data[15:0]	Out	VFBC read data
VFBC_rd_empty	In	VFBC read FIFO empty flag
VFBC_rd_almost_empty	In	VFBC read FIFO almost empty flag
VFBC_wr_clk	Out	VFBC write clock
VFBC_wr_reset	In	VFBC write reset
VFBC_wr_write	Out	VFBC write enable
VFBC_wr_end_burst	Out	VFBC write end burst
VFBC_wr_flush	Out	VFBC write flush
VFBC_wr_data_be[1:0]	Out	VFBC write data byte enables
VFBC_wr_data[15:0]	Out	VFBC write data
VFBC_wr_full	In	VFBC write FIFO full flag

Table 4: **VFBC and Genlock Signals *(Cont'd)***

Name	Direction	Description
VFBC_wr_almost_full	In	VFBC write FIFO almost full flag
VFBC_cmd_clk	Out	VFBC command FIFO clock
VFBC_cmd_reset	Out	VFBC command interface reset
VFBC_cmd_data[31:0]	Out	VFBC command data
VFBC_cmd_write	Out	VFBC command FIFO write enable
VFBC_cmd_end	Out	VFBC command end signal
VFBC_cmd_full	In	VFBC command FIFO full flag
VFBC_cmd_almost_full	In	VFBC command FIFO almost full flag
VFBC_cmd_idle	In	VFBC idle – indicates last command processed
rd_frame_ptr_in	In	Active video output indicates valid data on YCM interface (pCore only)

Table 5: **Processor Local Bus AXI4-Lite Signals**

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

Table 5: **Processor Local Bus AXI4-Lite Signals *(Cont'd)***

Name	Direction	Description
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.
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.
Sl_addAck	Out	Slave address acknowledge
Sl_SSize[0:1]	Out	Slave data bus size
Sl_wait	Out	Slave wait indicator
Sl_rearbitrate	Out	Slave rearbitrate bus indicator
Sl_wrDAck	Out	Slave write data acknowledge
Sl_wrComp	Out	Slave write transfer complete indicator
Sl_wrBTerm	Out	Slave terminate write burst transfer
Sl_rdDBus[0:C_SPLB_DWIDTH-1]	Out	Slave read data bus
Sl_rdWdAddr[0:3]	Out	Slave read word address
Sl_rdDAck	Out	Slave read data acknowledge
Sl_rdComp	Out	Slave read transfer complete indicator
Sl_rdBTerm	Out	Slave terminate read burst transfer

Table 5: **Processor Local Bus AXI4-Lite Signals *(Cont'd)***

Name	Direction	Description
SI_MBusy[0:C_SPLB_NUM_MASTERS-1]	Out	Slave busy indicator
SI_MrdErr[0:C_SPLB_NUM_MASTERS-1]	Out	Slave read error indicator
SI_MwrErr[0:C_SPLB_NUM_MASTERS-1]	Out	Slave write error indicator
SI_MIRQ[0:C_SPLB_NUM_MASTERS-1]	Out	Slave interrupt
IP2INTC_Irpt	Out	Interrupt signal

General Purpose Processor Interface

The other interface option is the General Purpose Processor (GPP) interface. The directly exposed control, interrupt and status signals allow the user to wrap these signals with a user-defined bus interface targeting any arbitrary processor. New values written to the control signals take effect immediately. See Tables 6 and 7.

Table 6: **GPP General Interface Signals**

Name	Direction	Description
sclr	In	System synchronous reset (active high)
clk	In	Main core clock
vsync_in	In	Vertical sync input
YCMOut_WE	Out	Active video output indicates valid data on YCMOut
chroma_valid_out	Out	Indicates validity of 4:2:0 chroma lines on alternate-line basis
YCMOut[23:0]	Out	Bits[7:0]: Luma out. Bits[15:8]: Chroma out. Bits[23:16]: Motion out

Table 7: **GPP Register Signals, VFBC Signals**

Name	Direction	Description
VFBC_rd_clk	Out	VFBC read clock
VFBC_rd_reset	Out	VFBC read data reset
VFBC_rd_read	Out	VFBC read data enable
VFBC_rd_end_burst	Out	VFBC read end burst
VFBC_rd_flush	Out	VFBC read flush
VFBC_rd_data[15:0]	Out	VFBC read data
VFBC_rd_empty	In	VFBC read FIFO empty flag
VFBC_rd_almost_empty	In	VFBC read FIFO almost empty flag
VFBC_wr_clk	Out	VFBC write clock
VFBC_wr_reset	In	VFBC write reset
VFBC_wr_write	Out	VFBC write enable
VFBC_wr_end_burst	Out	VFBC write end burst
VFBC_wr_flush	Out	VFBC write flush

Table 7: **GPP Register Signals, VFBC Signals**

Name	Direction	Description
VFBC_wr_data_be[1:0]	Out	VFBC write data byte enables
VFBC_wr_data[15:0]	Out	VFBC write data
VFBC_wr_full	In	VFBC write FIFO full flag
VFBC_wr_almost_full	In	VFBC write FIFO almost full flag
VFBC_cmd_clk	Out	VFBC command FIFO clock
VFBC_cmd_reset	Out	VFBC command interface reset
VFBC_cmd_data[31:0]	Out	VFBC command data
VFBC_cmd_write	Out	VFBC command FIFO write enable
VFBC_cmd_end	Out	VFBC command end signal
VFBC_cmd_full	In	VFBC command FIFO full flag
VFBC_cmd_almost_full	In	VFBC command FIFO almost full flag
VFBC_cmd_idle	In	VFBC idle – indicates last command processed
CurrFrame_Start_Addr[31:0]	Out	Starting address of current frame buffer
PrevFrame_Start_Addr[31:0]	In	Starting address of previous frame buffer
OutputFrame_Start_Addr[31:0]	In	Starting address of output frame buffer
VFBC_Stride[31:0]	Out	Amount of memory to allocate per line of video
active_line_length[11:0]	Out	Active line length register bits
active_frame_height[11:0]	Out	Active frame height register bits
MTF_Din[7:0]	Out	MTF data input
MTF_WE	Out	MTF data write enable
MTF_active_bank	In	MTF active bank select: '0' bank 1, '1' bank 2
MTF_wr_bank	In	MTF write bank select: '0' bank 1, '1' bank 2
MTF_wr_bank_we	In	MTF write bank write enable
MTF_Load_Done	In	MTF loading done: asserted when MTF has loaded
control[31:0]	In	Control register bits
framedone	Out	Indicates the end of the frame
MANR_vreset_Out	Out	General purpose reset signal asserted for one line period during vertical blanking
reg_update_done	Out	Issued during Vertical blanking when the register values have been transferred to the active registers
version	Out	Core HW version number

MANR Control and Timing

The initialization and startup of the MANR is very simple. After reset, the user need only initialize the registers as appropriate to set up the frame buffer addresses and other options. 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 5.

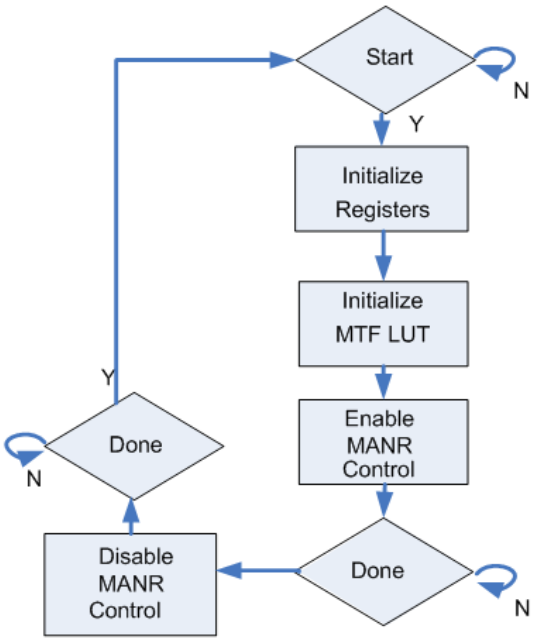

Figure 5: **MANR Initialization**

VFBC Interface Timing

The VFBC interface is the port through which the MANR reads and writes the video data for the current, output, and previous frames. Proper operation of this interface is crucial to the successful usage of the MANR core in any application.

The VFBC timing for the GPP and pCore are identical. The following is a brief description of how this interface operates.

- Following `vsync_in`, the MANR core starts to issue commands to the VFBC via the command interface signals `VFBC_cmd_write`, `VFBC_cmd_data(31:0)`.
- The commands stop being issued via this interface under one of two conditions:
  - The `VFBC_cmd_almost_full` goes high.
  - All read and write commands for the current frame have been issued.

- The MANR monitors the VFBC_cmd_idle signal to establish whether sufficient data is present in the VFBC read-data FIFO to start processing for the next line.
  - From the start of the frame to the end of the frame, the VFBC read-data FIFO must contain two complete lines (one from current frame and one from the previous frame) for the MANR to read it.
  - At the start of the frame, the MANR issues extra read commands to pre-fill the VFBC read-data FIFO.
  - This condition occurs once the VFBC_cmd_idle flag has occurred twice in this frame after vsync_in.
- Given the above conditions, the MANR asserts VFBC_rd_read (high) to read from the VFBC read FIFO.
  - The data on VFBC_rd_data is assumed to be active one cycle after VFBC_rd_read is asserted.
- The MANR reads two lines – one from the current frame, followed by the same line from the previous frame.
  - VFBC_rd_read is deasserted (low) after each line has been completely read into the MANR. Between the current and previous lines, this period will be at least 10 cycles.
  - To read the second (previous frame) line, the MANR also monitors the VFBC_wd_almost_full signal to establish that the VFBC write-data FIFO has sufficient empty capacity to accept a complete line from the MANR core.
- During the input of the line from the previous frame, the MANR also asserts VFBC_wd_write to write the MANR output line to the VFBC write-data FIFO via the data port VFBC_wd_data.

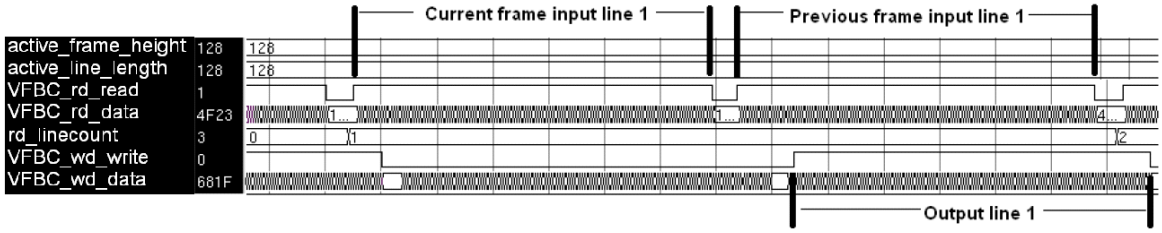

Figure 6: **VFBC Data Port Timing**

MTF Interface Timing

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

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

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

The MTF loading interface is an asynchronous interface. A high level on the MTF_WE signal is used to capture the MTF values delivered on MTF_Din. An internal state-machine detects the 3rd clock cycle when MTF_WE is stable and high. At this point, the data is registered into the internal MTF memory. Xilinx recommends that the MTF_WE pulse be no less than the equivalent of six clock periods in duration. It is also required that it be low for a period no less than six clock periods in duration between write operations. Figures 7 and 8 show the timing relationships for

the MTF interface. Figure 7 shows the detailed timing for two MTF values. Figure 8 shows the loading of all 64 values for the MTF.

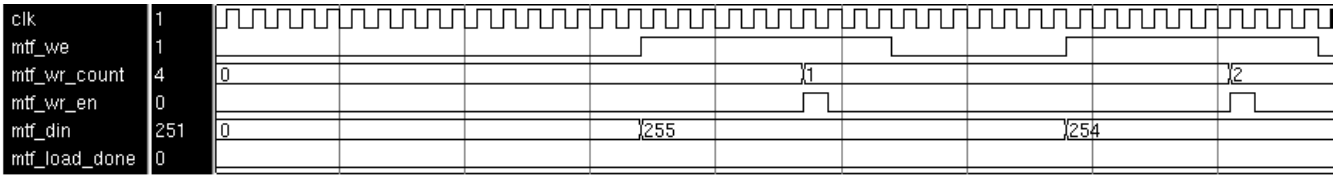

Figure 7: **MTF Port Timing**

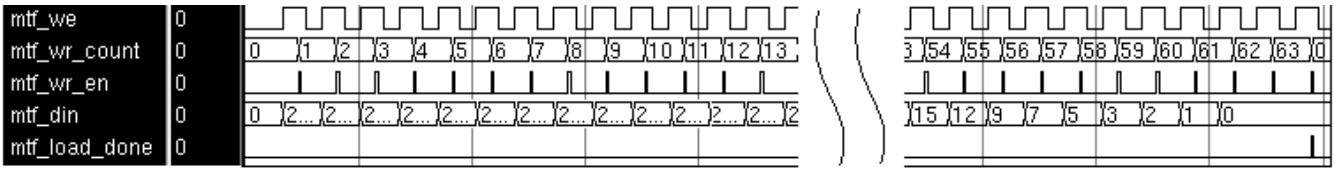

Figure 8: **MTF Load Timing**

YCM Interface Timing

The YCM interface is used to output Luma, Chroma, and motion data as a single 24-bit data word. This interface is for use with possible future Xilinx LogiCORE IPs, or for general processing of the video data without respect to video timing. A data valid signal is also provided as the `YCMOut_WE` signal. The timing of `YCMOut` and `YCMOut_WE` is identical to `VFBC_wd_data` and `VFBC_wd_write` as described in "VFBC Interface Timing."

MANR Frame Buffer Management

The MANR core generates the commands that control the VFBC. The commands that are sent to the VFBC include the start address of the line that is being accessed for each of the current (read), previous (read), and output (write) frames. For the GPP, these addresses are provided to the core via the `CurrFrame_Start_Addr[31:0]`, `PrevFrame_Start_Addr[31:0]`, and `OutputFrame_Start_ Addr[31:0]` ports.

For the pCore only, additional functionality has been included that allows the MANR to automatically rotate the buffer pointers within a 5-frame circular buffer in external memory. This provides genlock functionality and general frame read-write overtake protection that is often required in video systems. The pCore includes five frame buffer start address registers. The three GPP ports previously mentioned are selected from these register values according to the 5-bit gray-coded input port `rd_frame_ptr_in` which typically should be driven by an external genlock master. For reference, the Xilinx LogiCORE VDMA provides identical genlock functionality. More documentation can be found in the VDMA data sheet. The basic principles are outlined below.

Synchronous Circular Frame Buffer

For correct system functionality, it is generally important that the MANR core not overwrite any frame-data that is currently being written/read by some other function. Hence, part of the system design must include correct control of the three `Start_Addr` ports to avoid read/write collisions.

Use of a single rotating multi-frame buffer shared between all functional blocks is one way of achieving this. Conceptually, an area of memory subdivided into n single-frame buffer locations is allocated for this purpose. Successive functional system blocks access frame data from different frame pointers in this buffer. The number of frames of storage required (n) depends upon the number and function of distinct functional blocks in the system. Each frame buffer has a fixed start address in external memory. The most upstream block will write to the memory

in the order 1, 2… n, 1, 2, ... For a simple system whose multiple blocks are synchronized to a common vertical synchronization timing pulse (vsync), the next blocks that require frame data input may operate by accessing one (or more) of the ***completed*** frames in memory. Blocks that need to write a frame to memory must do so to a location that is not being read or written by other blocks. An example is given in Figure 9. In this case, n=4.

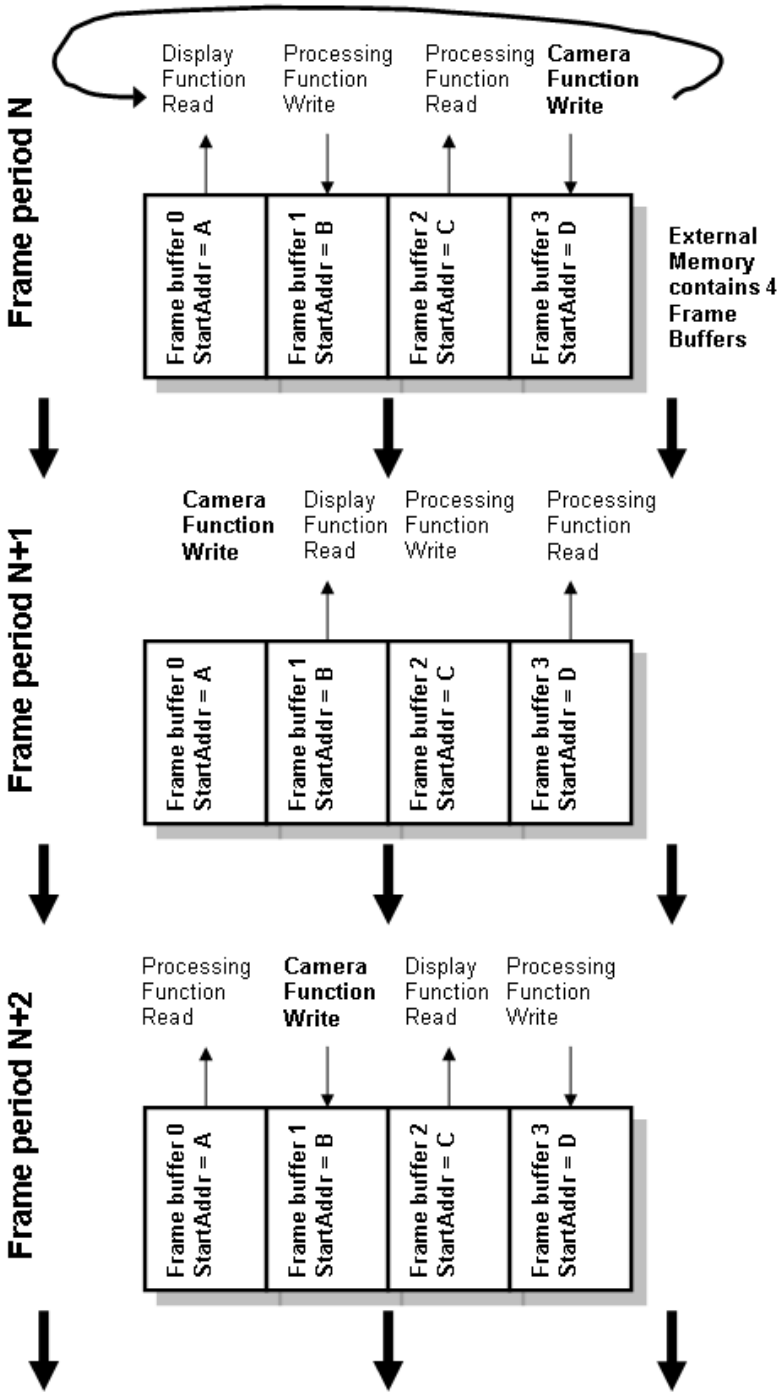

Figure 9: **Frame Circular Buffer Example**

The MANR core requires two input frames – current and previous – and one output frame. The output frame becomes the previous frame during the next frame period. Hence, in the next frame period, `PrevFrame_Start_Addr` is given the same value as is on `OutputFrame_Start_Addr` during the current frame period. The output frame is also the frame that is read by the next block. This is illustrated in Figure 10.

Because the MANR always writes a line output ***after*** having read the line from the two input frames, the output line may be written to the same location as the current frame (that is, during any frame period, `OutputFrame_Start_Addr` = `CurrFrame_Start_Addr`). This is also illustrated in Figure 10.

This mechanism allows the MANR to remain in sync with a synchronous video feed that rotates around a circular buffer arrangement as shown in Figure 9.

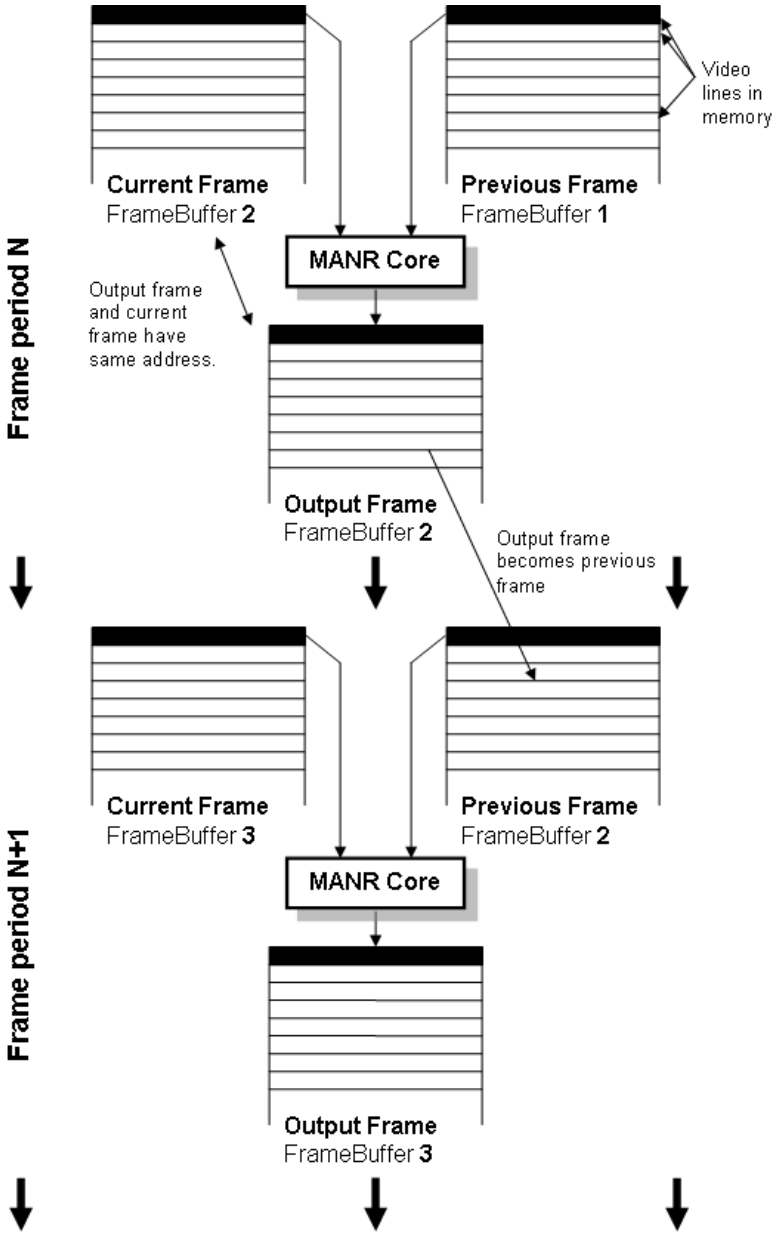

Figure 10: **MANR Operation within Circular Buffer**

Genlock

The previous scenarios assume that all frame buffer operations are synchronized. It is commonly the case that genlock functionality is required, however, because one of the frame buffer operations is occurring according to an asynchronous frame sync (vsync) and clock domain. Typically, the original source (for example, a camera) will provide one domain and the rest will be on the other domain. In this case, the camera domain typically becomes the genlock master. It informs the rest of the system which of the n frames it is writing, and the individual slaves decide on which frame to operate. Depending on the relationship between the clock domains, this will ultimately result in occasional frame skip or frame repeat. At least one more frame buffer is recommended for this purpose, especially in the case where the camera may be writing faster than the rest of the system. A 5-frame usage scenario is shown in Figure 11.

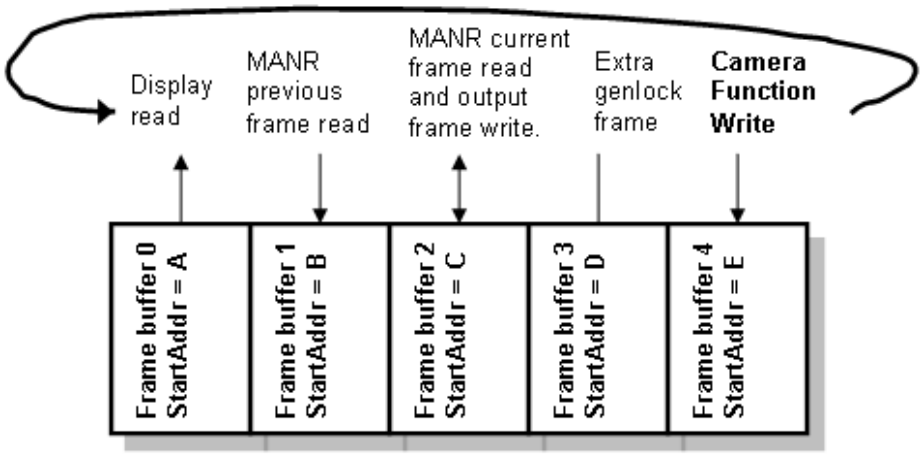

Figure 11: **Potential MANR 5-Frame Circular Buffer Usage**

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 the VFBC interface. 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 Figures 12 and 13, thick lines are used to indicate video data flow in the system.

Use Model 1: Noise Reduction Application

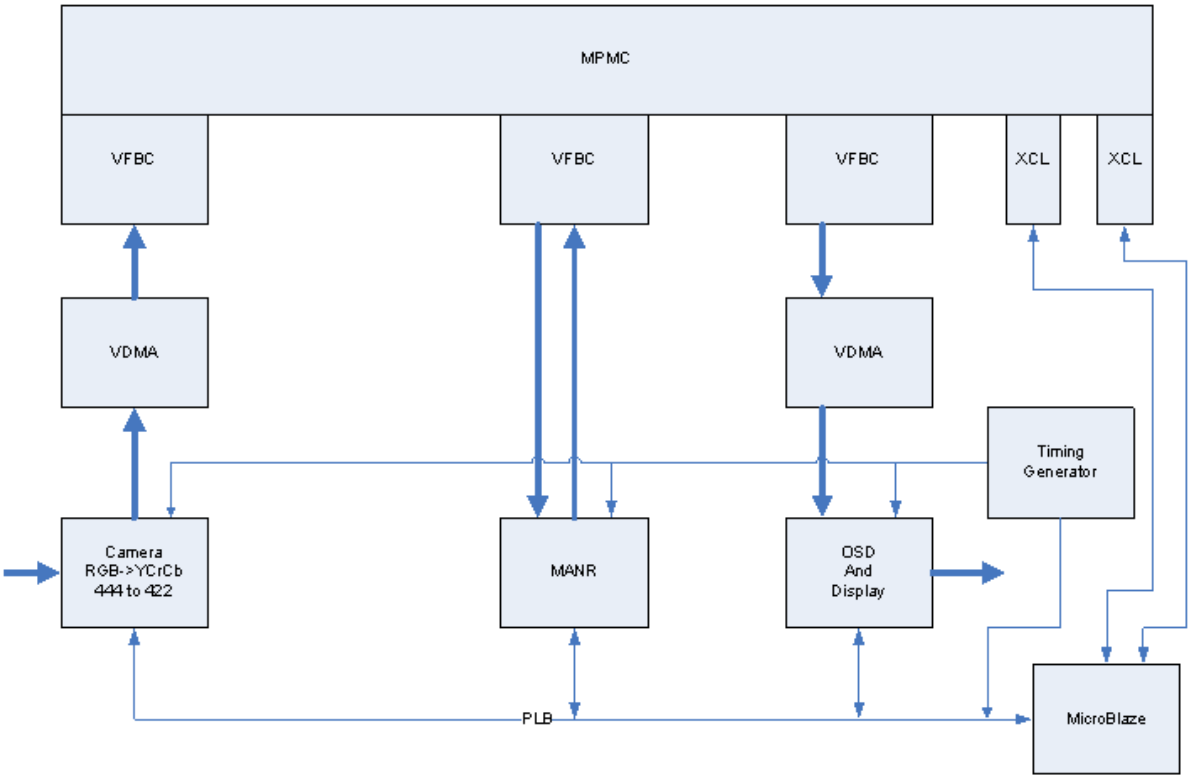

Figure 12: **Simple Noise Reduction**

Use Model 2: Noise Reduction and Motion Detection

In this example, the MANR is used to calculate and provide motion data and noise reduction in a simple video processing system. Such a system can be easily built using the building blocks provided by Xilinx (VDMA, Timing Controller, OSD, etc.)

In this application, the MANR core noise reduces the incoming video from the camera, and also provides the YCM data to a processing module via the YCM output port for additional processing of the motion and image data.

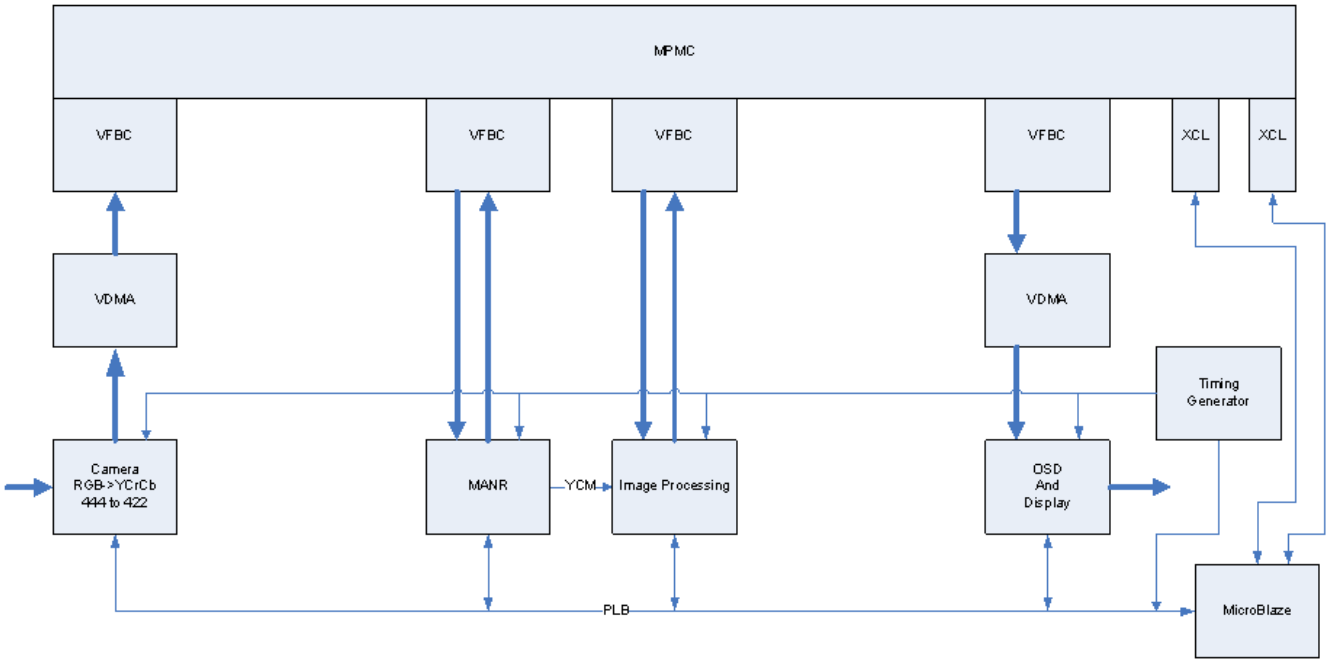

Figure 13: **Noise Reduction and Motion Processing**

Core Resource Utilization

Resources required for the MANR have been estimated for the Virtex®-5 (Table 8), Spartan®-3A DSP (Table 9), Virtex-6 (Table 10), and Spartan-6 (Table 11). These values were generated using the Xilinx CORE Generator tools v13.1. They are derived from post-synthesis reports, and may change during MAP and PAR. The Noise Reduction Strength setting has no effect on the core size. This setting changes only the initialized values of internal RAMs. Only the Max Frame Size setting has any effect on core size.

The pCore option affects only the LUT counts and F/F counts by a fixed delta. Add 1200 to the LUT and F/F numbers for pCore utilization numbers for Spartan3A-DSP; add 840 to the LUT number and 1200 to the F/F number for all other families.

Table 8: **Virtex-5 Utilization – GPP**

Configuration	LUTs	F/Fs	Block RAMs	DSP48E
640x360	559	835	1	3
640x480	565	837	1	3
720x480	563	837	1	3
1280x720	565	839	1	3
1920x1080	567	839	1	3

Table 9: **Spartan3A-DSP Utilization – GPP**

Configuration	LUTs	F/Fs	Block RAMs	DSP48A
640x360	773	836	1	3
640x480	775	838	1	3
720x480	775	838	1	3
1280x720	777	840	2	3
1920x1080	777	840	2	3

Table 10: **Virtex-6 Utilization – GPP**

Configuration	LUTs	F/Fs	Block RAMs	DSP48E1
640x360	563	825	1	3
640x480	565	827	1	3
720x480	565	827	1	3
1280x720	567	829	1	3
1920x1080	567	829	1	3

Table 11: **Spartan-6 Utilization - GPP**

Configuration	LUTs	F/Fs	Block RAMs	DSP48A1
640x360	565	833	1	3
640x480	567	835	1	3
720x480	567	835	1	3
1280x720	569	837	1	3
1920x1080	567	829	1	3

Performance

For the MANR to process a complete 720p60 or 1080p30 frame within one frame period, the internal clock must be run at 2X the pixel clock rate, or 150 MHz.

The following are typical clock frequencies for the target families. The maximum achievable clock frequency could vary and can be lower or higher. The maximum achievable clock frequency and all resource counts may be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools, and other factors.

- Spartan-3A DSP: 150 MHz
- Spartan-6: 150 MHz
- Virtex-5: 225 MHz
- Virtex-6: 250 MHz

Support

Xilinx provides technical support for this LogiCORE 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*.

License Options

The Xilinx Motion Adaptive Noise Reduction LogiCORE system provides three licensing options. After installing the required Xilinx ISE software and IP Service Packs, choose a license option:

Simulation Only

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

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the Motion Adaptive Noise Reduction core.

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

Full

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

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

Obtaining Your License Key

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

Simulation License

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

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license:

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

Obtaining a Full License

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

Installing Your License File

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

Ordering Information

The Motion Adaptive Noise Reduction v2.0 core is provided under the SignOnce IP Site License and can be generated using the Xilinx CORE Generator system v13.1 or higher. The CORE Generator system is shipped with Xilinx ISE Design Suite development software.

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

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
12/02/09	1.0	Initial Xilinx release.
09/21/10	1.1	Updated for 12.3 release.
03/01/11	2.0	Updated for 13.1 release.

Notice of Disclaimer