# Motion Adaptive Noise Reduction v6.0

## *Product Guide for Vivado Design Suite*

**PG006 October 2, 2013**

# Table of Contents

## 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 Vivado IP catalog 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.
  - Supports user-defined MTF functions.
- Full support for interrupts and status registers for easy system control.
- Supports YUV 4:2:2 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 [1]

1. Performance on low power devices may be lower.

| LogiCORE IP Facts Table | |
|---|---|
| **Core Specifics** | |
| Supported Device Family[1] | Zynq®-7000, Artix®-7, Virtex®-7, Kintex®-7 |
| Supported User Interfaces | AXI4-Lite, AXI4-Stream [2] |
| Resources | See Table 2-1 through Table 2-4. |
| **Provided with Core** | |
| Documentation | Product Guide |
| Design Files | Encrypted RTL |
| Example Design | Not Provided |
| Test Bench | Verilog |
| Constraints File | XDC |
| Simulation Models | Encrypted RTL, VHDL or Verilog Structural, C-Model |
| Supported Software Drivers [3] | Standalone |
| **Tested Design Tools** | |
| Design Entry Tools | Vivado® Design Suite IP Integrator |
| Simulation[4] | For supported simulators, see the Xilinx Design Tools: Release Notes Guide. |
| Synthesis Tools | Vivado Synthesis |
| **Support** | |
| Provided by Xilinx, Inc. | |

1. For a complete listing of supported devices, see the Vivado IP Catalog.
2. Video Protocol as defined in UG761, *Xilinx AXI Reference Guide* [Ref 3].
3. Standalone driver details can be found in the SDK directory (<*install_directory*>/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from //wiki.xilinx.com.
4. For the supported versions of the tools, see the Xilinx Design Tools: 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 combining the current pixel values and a percentage of the previous pixel values. Large changes in pixel values between successive frames likely indicate motion, and should be preserved in the output frame. Smaller changes are more likely caused by noise in the current frame, therefore averaging with pixel values from the previous frame can be applied to suppress noise. This recursive action effectively reduces noise while preserving the output image content by masking small changes but preserving larger pixel changes.

The core uses pixel values from the current and previous frames as inputs from which temporal differences are established. A two-dimensional filtering kernel is used to analyze inter-frame differences. Averaged over a 3x5 area, random noise induces smaller changes than objects moving in or out of the kernel area. The absolute value of the averaging filter output is used as an argument to a grading function, the motion-transfer-function (MTF). The function value is used as a blending factor between the current pixel value and the previous pixel value. This temporal IIR structure allows the core to generate optimal output values where temporal noise is reduced, but motion is conserved.

The grading, or MTF is programmable in the MANR core. Typically, the function maps the smaller pixel-differences, likely to be noise, to large blending factor values. Conversely, larger pixel-differences, likely to be motion, are mapped to small blending factor values. The blending factor controls what portion of an output pixel comes from the previous frame, and what is carried forward from the current frame.

## Feature Summary

The MANR core supports resolutions of up to 4096x4096 using 8-bit YC4:2:2 chroma formats. The Vivado IP catalog GUI provides five preset MANR strengths which correspond to preset MTFs. The core also supports overloading any of the 16 motion transfer functions with user-defined values.

The MANR core uses two AXI4-Stream input (slave) interfaces in parallel: one for the current frame and one for the previous frame. Typically, the current frame is provided from a live source or an AXI VDMA. The previous frame typically originates from a frame buffer, usually connected to the AXI VDMA core. The core processes 8-bit video data in YCC 422 format only.

To dynamically change the noise reduction strength on a frame-by-frame basis, a processor interface is required. When generating the MANR core, you have the option of including a processor interface that is instantiated in the core.

## Applications

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

## Licensing and Ordering Information

This Xilinx® LogiCORE™ IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability.

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

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

# Product Specification

The Motion Adaptive Noise Reduction IP core an be used as either a stand-alone core or as peripheral to a processor system. An optional AXI4-Lite interface with user registers, interrupts and device driver make the Motion Adaptive Noise Reduction module highly programmable and easy to control in real-time with a processor.

## Standards Compliance

The MANR core is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. See *Video IP: AXI Feature Adoption* section of the *AXI Reference Guide (UG761)*[Ref 3] for additional information.

## Performance

The following sections detail the performance characteristics of the MANR core.

### Maximum Frequency

The resource utilization tables contain typical clock frequencies for the target devices. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools and other factors. Refer to in Table 2-1 through Table 2-4 for device-specific information. These results are typical, and have been used as target clock frequencies for the MANR in the slowest speed-grade for each device family, and apply to the main operation clock signal `aclk`.

### Latency

The latency through the MANR core is at least one full scan line and 17 pixels. 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 core supports bidirectional data throttling between the AXI4-Stream Slave and Master interfaces. If the slave-side data sources are not providing valid data samples (because `s_axis_currframe_tvalid` or `s_axis_prevframe_tvalid` is not asserted), the core cannot produce valid output samples after the internal buffers are depleted. Similarly, if the master-side interface is not ready to accept valid data samples (because `m_axis_mem_tready`, `m_axis_output_tready` or `m_axis_motion_tready` is not asserted) the core cannot accept valid input samples once the buffers become full.

If the master interfaces are able to provide valid samples and the slave interface are ready to accept valid samples, the core can process one sample and produce one pixel per `aclk` cycle.

However, at the end of each scanline, the core flushes internal pipelines for approximately 20 clock cycles, during which the `s_axis_currframe_tready` and `s_axis_prevframe_tready` are de-asserted signaling that the core is not ready to process samples. Similarly, at the end of the frame, after the last expected scanline is completed (EOL received), the core flushes internal pipelines for approximately 1 line-time.

When the core is processing timed-streaming video, the flushing periods coincide with the blanking periods, and do not reduce the throughput of the system. If timing constraints are met, the throughput is equal to the rate at which video data is written into the core. 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. To ensure that the core can process 1080P/60 video, the core must run at least 130 MHz when input buffers can accommodate a full line, or 148.5 MHz in all other cases.

# Resource Utilization

To help guide you in making system-level and board-level decisions, Table 2-1 through Table 2-4 show the resource usage observed for the MANR for all supported devices in either Vivado™ Design Suite. This post-implementation characterization data has been collated through automated implementation of each configuration. Results may vary between implementations, and are intended as guidelines. Resource count and Fmax values are independent of the actual video resolution (but dependent on maximum supported resolutions), as well as the noise reduction value selected in the GUI. The results were generated by setting the maximum number of active pixels per line to 1920, maximum number of active lines per frame to 1080, and not selecting the INTC Interface option. The core does not use any 9k block RAM, or dedicated clock, routing, or I/O devices.

***Note:*** Performance (FMax) on Zynq-7020, Zynq-7010 and low-power (-L) devices may be lower than quoted in Table 2-1 through Table 2-4.

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

| AXI4-Lite | LUTs | FFs | BRAM36 | BRAM18 | DSP48 | FMax (MHz) Per Speed Grade | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | -1 | -2 | -3 |
| No | 717 | 895 | 3 | 1 | 2 | 266 | 312 | 344 |
| Yes | 1163 | 1487 | 3 | 1 | 2 | 242 | 288 | 312 |

**Notes:**
1. Speedfile: PRELIMINARY 1.05 2013-02-09

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

| AXI4-Lite | LUTs | FFs | BRAM36s | BRAM18s | DSP48s | FMax (MHz) Per Speed Grade | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | -1 | -2 | -3 |
| No | 719 | 895 | 3 | 1 | 2 | 258 | 328 | 360 |
| Yes | 1167 | 1487 | 3 | 1 | 2 | 234 | 296 | 328 |

**Notes:**
1. Speedfile: PRODUCTION 1.09c 2013-02-09

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

| AXI4-Lite | LUTs | FFs | BRAM36s | BRAM18s | DSP48s | FMax (MHz) Per Speed Grade | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | -1 | -2 | -3 |
| No | 719 | 895 | 3 | 1 | 2 | 250 | 304 | 375 |
| Yes | 1165 | 1487 | 3 | 1 | 2 | 226 | 274 | 320 |

**Notes:**
1. Speedfile: PRODUCTION 1.08 2013-02-09

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

| AXI4-Lite | LUTs | FFs | BRAM36s | BRAM18s | DSP48s | FMax (MHz) Per Speed Grade | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | -1 | -2 | -3 |
| No | 719 | 895 | 3 | 1 | 2 | 172 | 212 | 226 |
| Yes | 1162 | 1487 | 3 | 1 | 2 | 156 | 204 | 226 |

**Notes:**
1. Speedfile: PRODUCTION 1.08b 2013-02-09

## 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 MANR 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 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.

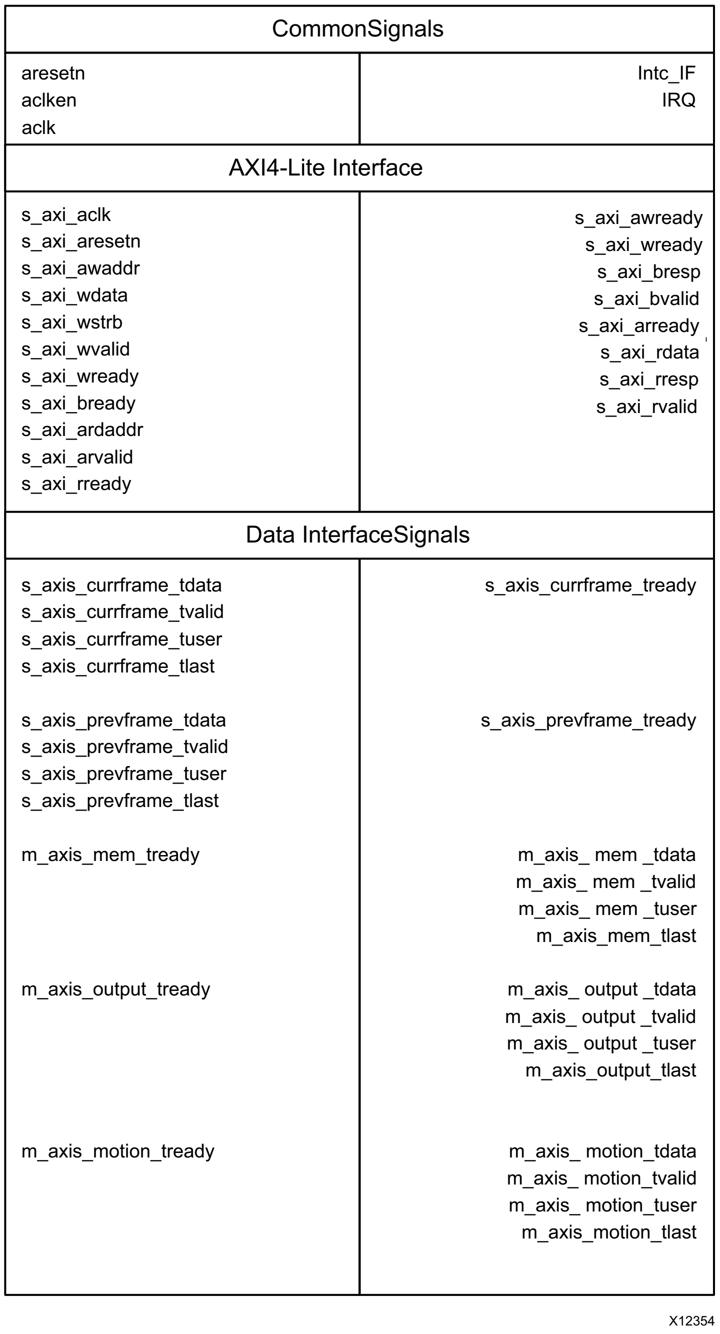


*Figure 2-1:* **MANR Interfaces and Signals**

## Common Interface Signals

Table 2-5 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

*Table 2-5:* **Common Interface Signals**

| Signal Name | Direction | Width | Description |
|---|---|---|---|
| aclk | In | 1 | Video core clock |
| aresetn | In | 1 | Video core active low synchronous reset |
| aclken | In | 1 | Video core clock enable |
| intc_if | Out | 14 | Optional external interrupt controller interface.<br>Available only when INTC_IF is selected in the GUI. |
| irq | Out | 1 | Optional interrupt request pin.<br>Available only when INTC_IF is selected in the GUI. |

The `aclk`, `aclken`, and `aresetn` signals are shared between the core and the AXI4-Stream data interfaces. The AXI4-Lite control interface has its own set of clock and reset pins: `s_axi_aclk` and `s_axi_aresetn`. See Interrupt Subsystem for a description of the `intc_if` and `irq` pins.

### ACLK

The AXI4-Stream interface must be synchronous to the core clock signal `aclk`. All AXI4-Stream interface input signals are sampled on the rising edge of `aclk`. All AXI4-Stream output signal changes occur after the rising edge of `aclk`. The AXI4-Lite interface is unaffected by the `aclk` signal.

### ARESETn

The `aresetn` pin is an active-low, synchronous reset input pertaining to only AXI4-Stream interfaces. The core resets on the next rising `aclk` edge after `aresetn` is asserted low. The `aresetn` signal must be synchronous to the `aclk` and must be held low for a minimum of 32 clock cycles of the slowest clock. The AXI4-Lite interface is unaffected by the `aresetn` signal.

### ACLKEN

The `aclken` pin is an active high pin which can enable or disable AXI4-Streams and core processing functionality on a clock cycle by clock cycle basis. The `aclken` pin facilitates building multi-cycle path, or multi-rate designs.

## Data Interface

The core receives and transmits data using AXI4-Stream interfaces that implement a video protocol as defined in *Xilinx AXI Reference Guide (UG761)*[Ref 3], Video IP: "AXI Feature Adoption."

### AXI4-Stream Signal Names and Descriptions

Table 2-6 describes the AXI4-Stream signal names and descriptions

*Table 2-6:* **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 |
| 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 |

*Table 2-6:* **AXI4-Stream Data Interface Signal Descriptions *(Cont'd)***

| Signal Name | Direction | Width | Description |
|---|---|---|---|
| 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 processes 8-bit YCC 422 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`, 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.

***Note:*** When interfacing to an AXI4-Stream master interface using an `aclken` input, which is not permanently tied high, the two interfaces must be connected using the AXI4 FIFO core to avoid data corruption. See the *LogiCORE IP FIFO Generator Product Guide (PG057)*[Ref 2].

## Guidelines on Driving VALID into Slave (Data Input) Interfaces

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

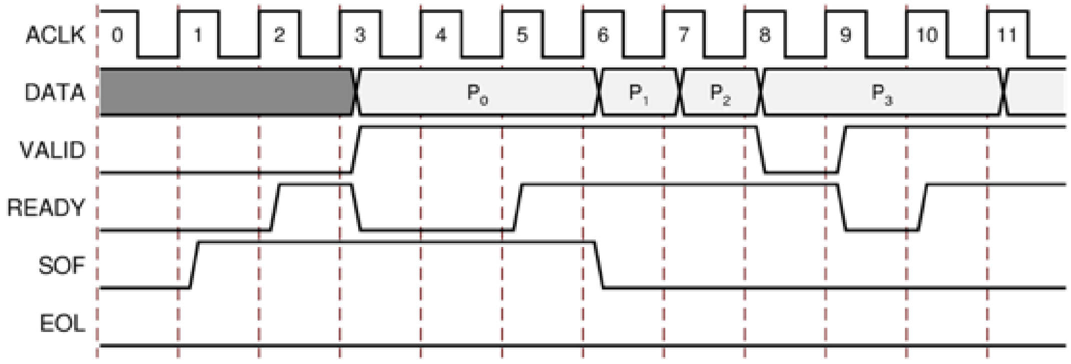


*Figure 2-2:* **Example of READY/VALID Handshake, Start of a New Frame**

### Guidelines on Driving READY into Master (Data Output) Interfaces

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

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

### Start of Frame Signals: m_axis_video_tuser, s_axis_video_tuser

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 VALID is not asserted.

The MANR core synchronizes the inputs so that SOF signals on the axis_currframe and the axis_prevframe inputs coincide. Synchronization takes place by de-asserting the axis_prevframe_tready signal if SOF is detected at the output of the internal FIFO buffering the axis_prevframe interface.

At the same time, axis_currframe_tready is asserted, but samples from the axis_currframe are dropped until SOF on the buffer output is sampled. Subsequently normal data processing can commence.

### End of Line Signals: m_axis_video_tlast, s_axis_video_tlast

The End-Of-Line (EOL) 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.

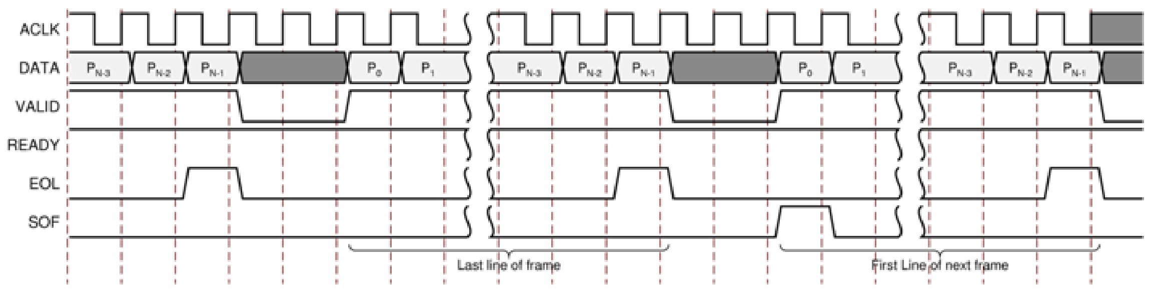


*Figure 2-3:* **Use of EOL and SOF Signals**

## 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 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 2-7.

*Table 2-7:* **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.<br>1 = Write address is valid.<br>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.<br>1 = Ready to accept address.<br>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.<br>1 = Write data/strobes are valid.<br>0 = Write data/strobes are not valid. |

*Table 2-7:* **AXI4-Lite Control Bus Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| s_axi_wready | Out | AXI4-Lite Write Data Channel Write Data Ready. Indicates core is ready to accept the write data.<br>1 = Ready to accept data.<br>0 = Not ready to accept data. |
| s_axi_bresp | Out | AXI4-Lite Write Response Channel. Indicates results of the write transfer.<br>00b = OKAY - Normal access has been successful.<br>01b = EXOKAY - Not supported.<br>10b = SLVERR - Error.<br>11b = DECERR - Not supported. |
| s_axi_bvalid | Out | AXI4-Lite Write Response Channel Response Valid. Indicates response is valid.<br>1 = Response is valid.<br>0 = Response is not valid. |
| s_axi_bready | In | AXI4-Lite Write Response Channel Ready. Indicates Master is ready to receive response.<br>1 = Ready to receive response.<br>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.<br>1 = Read address is valid.<br>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.<br>1 = Ready to accept address.<br>0 = Not ready to accept address. |
| s_axi_rdata | Out | AXI4-Lite Read Data Bus |
| s_axi_rresp | Out | AXI4-Lite Read Response Channel Response. Indicates results of the read transfer.<br>00b = OKAY - Normal access has been successful.<br>01b = EXOKAY - Not supported.<br>10b = SLVERR - Error.<br>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.<br>1 = Read data is valid.<br>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.<br>1 = Ready to accept data.<br>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 2-8.

*Table 2-8:* **MANR Registers**

| Address (hex) BASEADDR + | Register Name | Access Type | Double Buffered | Default Value | Register Description |
|---|---|---|---|---|---|
| 0x0000 | CONTROL | R/W | No | 0 | b0: SW_ENABLE<br>b1: REG_UPDATE<br>b2: MTF_BYPASS<br>b30: FRAME_SYNC_RESET (1: reset)<br>b31: SW_RESET (1: reset) |
| 0x0004 | STATUS | R/W | No | 0 | b0: FRAME_STARTED<br>b1: FRAME_COMPLETE<br>b2: AXI4_SLAVE_ERROR<br>b3: MTF_LOAD_DONE |
| 0x0008 | ERROR | R/W | No | 0 | b0: CURR_EOL_EARLY<br>b1: CURR_EOL_LATE<br>b2: CURR_SOF_EARLY<br>b3: CURR_SOF_LATE<br>b4: PREV_EOL_EARLY<br>b5: PREV_EOL_LATE<br>b6: PREV_SOF_EARLY<br>b7: PREV_SOF_LATE<br>b8: PIXEL_CNT_TC<br>b9: LINE_CNT_TC |
| 0x000C | IRQ_ENABLE | R/W | No | 0 | b0: MTF_LOAD_DONE_IRQEN<br>b1: FRAME_STARTED_IRQ_EN<br>b2: AXI4_SLAVE_ERROR_IRQ_EN |
| 0x0010 | Version | R | No | 0x06000000 | 15-0: Reserved<br>23-16: VERSION_MINOR<br>31-24: VERSION_MAJOR |
| 0x0014 | SYSDEBUG0 | R | N/A | 0 | 31-0: Frame Throughput monitor |
| 0x0018 | SYSDEBUG1 | R | N/A | 0 | 31-0: Line Throughput monitor |
| 0x001C | SYSDEBUG2 | R | N/A | 0 | 31-0: Pixel Throughput monitor |
| 0x0020 | ACTIVE_SIZE | R/W | Yes | Specified in GUI | 12-0: Number of active pixels per line<br>28-16: Number of active lines per frame |
| 0x0100 | MTF_DIn | W | Yes | 0 | 7-0: MTF Input Data. |

*Table 2-8:* **MANR Registers *(Cont'd)***

| Address (hex) BASEADDR + | Register Name | Access Type | Double Buffered | Default Value | Register Description |
|---|---|---|---|---|---|
| 0x0104 | MTF_Active | R/W | Yes | Specified in GUI | 3-0: Internal MTF Bank currently in use by MANR core |
| 0x0108 | MTF_Write | R/W | Yes | 0 | 3-0: Internal MTF Bank currently written by the processor interface (if present) |
| 0x010C | Frame_Noise | R | N/A | 0 | Estimated Noise content of the last frame processed |
| 0x0110 | Frame_Motion | R | N/A | 0 | Global Estimated Motion content of the last frame processed |

## 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 semaphore for the host processor, which facilitates committing all updates to double-buffered user and timing registers simultaneously. 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 at the start of a new frame, indicated by the SOF signal. 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 turned off.
- Bits 30 and 31 of the `CONTROL` register, `FRAME_SYNC_RESET` and `SW_RESET` facilitate 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 causes image tearing. For applications where the reset functionality is desirable, but image tearing has to be avoided, a frame synchronized reset (`FRAME_SYNC_RESET`) is available.

- Setting FRAME_SYNC_RESET to 1 resets the core at the end of the frame being processed, or immediately if the core is between frames when the FRAME_SYNC_RESET was asserted. After reset, the FRAME_SYNC_RESET bit is automatically cleared, so the core can get ready to process the next frame of video as soon as possible.
- The default value of both RESET bits is 0. Core instances with no AXI4-Lite control interface can only be reset through the aresetn pin.

### STATUS (0x0004) Register

Bits of the STATUS register can be used to request an interrupt from the host processor. 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, FRAME_STARTED signals the processor that the core started processing a new video frame. This bit is set when a valid START_OF_FRAME signal was registered on s_axis_currframe_tuser(0). If any user or timing register values were changed, this bit also signals that the core started using the set of register updates recently committed by REG_UPDATE (Bit 1 of the CONTROL register).
- Bit 1 of the STATUS register, FRAME_COMPLETE indicates that processing of a frame has completed. This bit is asserted when the core received all the pixel lines as programmed by the FRAME_SIZE (0x0100) register.
- Bit 2 of the STATUS register, AXI4_SLAVE_ERROR indicates that one of the bits of the ERROR registers were set by an AXI4-Stream error either on the previous or current frame inputs.
- Bit 3 of the STATUS register, MTF_LOAD_DONE, should be used when loading MTF values into the core. Following a successful transfer of 128 MTF values, the core asserts the MTF_LOAD_DONE status bit. For more information, see Motion Transfer Function (MTF) Access and Programming.

## ERROR (0x0008) Register

Table 2-9 describes the ERROR register bit.

*Table 2-9:* **Error Register Bit Functions**

| ERROR Register Bit | Name | Function |
|---|---|---|
| 0 | CURR_EOL_EARLY | Indicates that s_axis_currframe_tlast was asserted too early with respect to the values configured in the FRAME_SIZE register.<br>The expected position of this pulse is defined according to the source video resolution settings. |
| 1 | CURR_EOL_LATE | Indicates that s_axis_currframe_tlast was asserted<br>too late considering the previous TLAST pulse and the to the values configured in the FRAME_SIZE register. |
| 2 | CURR_SOF_EARLY | Indicates that s_axis_currframe_tuser was asserted too early with respect to the values configured in the FRAME_SIZE register.<br>The expected position of this pulse is defined according to the source video resolution settings. |
| 3 | CURR_SOF_LATE | Indicates that s_axis_currframe_sof was asserted too late considering the previous SOF pulse and the to the values configured in the FRAME_SIZE register. |
| 4 | PREV_EOL_EARLY | Indicates that s_axis_prevframe_tlast was asserted too early with respect to the values configured in the FRAME_SIZE register.<br>The expected position of this pulse is defined according to the source video resolution settings. |
| 5 | PREV_EOL_LATE | Indicates that s_axis_prevframe_tlast was asserted<br>too late considering the previous TLAST pulse and the to the values configured in the FRAME_SIZE register. |
| 6 | PREV_SOF_EARLY | Indicates that s_axis_prevframe_tuser was asserted too early with respect to the values configured in the FRAME_SIZE register.<br>The expected position of this pulse is defined according to the source video resolution settings. |
| 7 | PREV_SOF_LATE | Indicates that s_axis_prevframe_sof was asserted too late considering the previous SOF pulse and the to the values configured in the FRAME_SIZE register. |
| 8 | PIXEL_CNT_TC | Indicates that the number of pixels per lines measured (number of valid pixels received between EOL pulses) is larger than the Horizontal Size specified in the Frame Maximum dimensions panel of the GUI. |
| 9 | LINE_CNT_TC | Indicates that the number of lines per frame measured (number of EOL pulses between SOF pulses) is larger than the Vertical Size specified in the Frame Maximum dimensions panel of the GUI. |

Bits of the ERROR register can be used to request an interrupt from the host processor indirectly. If any of the bits of the ERROR register transitions high, Bit 2 of the STATUS register (AXI4_SLAVE_ERROR) is also asserted. If the corresponding bit (Bit 2) of the

IRQ_ENABLE register is set, the event causes the IRQ pin to transition high, and remain high until the interrupt source in the ERROR register is cleared.

To facilitate identification of the interrupt source, bits of the ERROR 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 ERROR register can be cleared individually by writing '1' to the bit position to be cleared.

## IRQ_ENABLE (0x000C) Register

Bits 0-3 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 register are masked by (AND) corresponding bits of the IRQ_ENABLE register. 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.

## SYSDEBUG0 (0x0014) Register

The SYSDEBUG0, or Frame Throughput Monitor, register indicates the number of frames processed since power-up or the last time the core was reset. The SYSDEBUG registers can be useful to identify external memory / Frame buffer / or throughput bottlenecks in a video system. Refer to Debug Tools in Appendix C for more information.

## SYSDEBUG1 (0x0018) Register

The SYSDEBUG1, or Line Throughput Monitor, register indicates the number of lines processed since power-up or the last time the core was reset. The SYSDEBUG registers can be useful to identify external memory / Frame buffer / or throughput bottlenecks in a video system. Refer to Debug Tools in Appendix C for more information.

## SYSDEBUG2 (0x001C) Register

The SYSDEBUG2, or Pixel Throughput Monitor, register indicates the number of pixels processed since power-up or the last time the core was reset. The SYSDEBUG registers can be useful to identify external memory / Frame buffer / or throughput bottlenecks in a video system. Refer to Debug Tools in Appendix C for more information.

## ACTIVE_SIZE (0x0020) Register

The `ACTIVE_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. To avoid processing errors, restrict values written to `ACTIVE_SIZE` to the range supported by the core instance.

## Motion Transfer Function (MTF) Access and Programming

Registers `MTF_DIn` (0x0100), `MTF_Active` (0x0104) and `MTF_Write` (0x0108) allow access to and programming of 16 programmable MTF locations. MTFs establish a relationship between pixel value changes between the current and the previous frames, and the temporal low-pass filtering that occurs at each pixel location. For best results, the MTF should be matched to noise and motion content of the video sequence being processed.

The MANR core can store 16 MTFs simultaneously, each defined by 128 words of data, the first 64 words for the Luma MTF, and the second 64 words for the Chroma MTF. The core uses the MTF bank identified by bits 3-0 of register `MTF_Active`. The first 8 MTF banks come pre-configured, with bank 0 pertaining to the MTF matched to a scene with little motion and heavy noise load, address 7 pertaining to the MTF matched to a scene with lots of motion and small noise load.

If none of the predefined Motion Transfer Functions fit the application, you can download custom MTFs to any of the 16 banks. The bank currently being written by the SW application is selected by bits 3-0 of the `MTF_Write` register. To avoid image tearing, `MTF_Write` and `MTF_Active` should select different MTF banks.

Load the desired motion transfer function into the Motion Transfer bank through the `MTF_DIn` port. Loading the MTF involves writing 128 8-bit unsigned coefficient values within the range 0 through 255, specifying fixed point values in the 0.0-0.996 range. A coefficient of 0 specifies propagating the current value forward. Refer to Chapter 3, Designing with the Core for more information.

The 128 values must be loaded sequentially, starting at element 0. Writing any value to the `MTF_Write` register resets the internal MTF loading process. 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 selecting the target MTF bank by writing `MTF_Write` (0x0108) register, then sequentially writing 128 values into the `MTF_DIn` (0x0100) register.

### Matching MTFs to Scene Content: The Frame_Noise and Frame_Motion Registers

To help dynamically select the MTF best fitting the scenario, the core reports the measured global motion and global noise content pertaining to the last frame processed. The values provided are not normalized per pixel, instead represent the frame as a whole. Thus, for the same noise variance and motion content, the larger the frame resolution, the larger the resulting register values becomes.

Separation of motion and noise is controlled by the current MTF selected. Pixel variations below the MTF= 0.5 are considered noise, above MTF=0.5 are considered motion. For two identical consecutive frames, both indicators return 0. For two consecutive frames with no motion but low variance noise, only the `frame_noise` register should return a non-zero value.

## 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, you can still benefit from the core signaling error and status events. By selecting the **Enable INTC Port** option, the core generates the optional `intc_if` port. This vector of signals gives parallel access to the individual interrupt sources, as shown in Table 2-10. Unlike `STATUS` and `ERROR` flags, `intc_if` signals are not held. They stay asserted only while the corresponding event persists.

*Table 2-10:* **INTC_IF Signal Functions**

| INTC_IF Signal | Name | Function |
|---|---|---|
| 0 | FRAME_STARTED | Indicates that the core has started processing a new frame. |
| 1 | FRAME_COMPLETE | Indicates that processing of a frame has completed. This bit is asserted when the core received all the pixel lines as programmed by the FRAME_SIZE (0x0100) register. |
| 2 | AXI4_SLAVE_ERROR | Indicates a framing error on either the previous or current frame AXI4-Stream slave (input) interfaces. |

*Table 2-10:* **INTC_IF Signal Functions *(Cont'd)***

| INTC_IF Signal | Name | Function |
|---|---|---|
| 3 | MTF_LOAD_DONE | Following a successful transfer of 128 MTF values, the core asserts the `MTF_LOAD_DONE` status bit. |
| 4 | CURR_EOL_EARLY | Indicates that s_axis_currframe_tlast was asserted too early with respect to the values configured in the FRAME_SIZE register.<br>The expected position of this pulse is defined according to the source video resolution settings. |
| 5 | CURR_EOL_LATE | Indicates that s_axis_currframe_tlast was asserted<br>too late considering the previous TLAST pulse and the to the values configured in the FRAME_SIZE register. |
| 6 | CURR_SOF_EARLY | Indicates that s_axis_currframe_tuser was asserted too early with respect to the values configured in the FRAME_SIZE register.<br>The expected position of this pulse is defined according to the source video resolution settings. |
| 6 | CURR_SOF_LATE | Indicates that s_axis_currframe_sof was asserted too late considering the previous SOF pulse and the to the values configured in the FRAME_SIZE register. |
| 8 | PREV_EOL_EARLY | Indicates that s_axis_prevframe_tlast was asserted too early with respect to the values configured in the FRAME_SIZE register.<br>The expected position of this pulse is defined according to the source video resolution settings. |
| 9 | PREV_EOL_LATE | Indicates that s_axis_prevframe_tlast was asserted<br>too late considering the previous TLAST pulse and the to the values configured in the FRAME_SIZE register. |
| 10 | PREV_SOF_EARLY | Indicates that s_axis_prevframe_tuser was asserted too early with respect to the values configured in the FRAME_SIZE register.<br>The expected position of this pulse is defined according to the source video resolution settings. |
| 11 | PREV_SOF_LATE | Indicates that s_axis_prevframe_sof was asserted too late considering the previous SOF pulse and the to the values configured in the FRAME_SIZE register. |
| 12 | PIXEL_CNT_TC | Indicates that the number of pixels per lines measured (number of valid pixels received between EOL pulses) is larger than the Horizontal Size specified in the Frame Maximum dimensions panel of the GUI. |
| 13 | LINE_CNT_TC | Indicates that the number of lines per frame measured (number of EOL pulses between SOF pulses) is larger than the Vertical Size specified in the Frame Maximum dimensions panel of the GUI. |

In a system integration tool, 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, you can custom build a priority interrupt controller to aggregate interrupt requests and identify interrupt sources.

# 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.

First, the difference between the current and previous frames are calculated. The difference image is filtered by a 2D averaging FIR filter, using a 3x5 kernel. The absolute value of the low-pass filter output, the motion value, is affected to a greater extent by objects moving in or out of the kernel (motion) than random noise.

This motion value is used as an index to the MTF look-up table. The Motion Transfer Function reflects the probability density function of the noise superimposed on the stream. Assuming Gaussian noise, the S-curve shape of the MTF intends to minimize the error that motion is falsely characterized as noise.

Second, the value corresponding to the calculated motion value from the MTF is used as a multiplier to scale the pixel-by-pixel 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 3-1.

For the MANR core, the above operation is carried out independently for luma and chroma channels. Separate engines are included for each channel. The sub-sampled Cr and Cb channels use this second engine. Switching between Cr and Cb is handled internally.

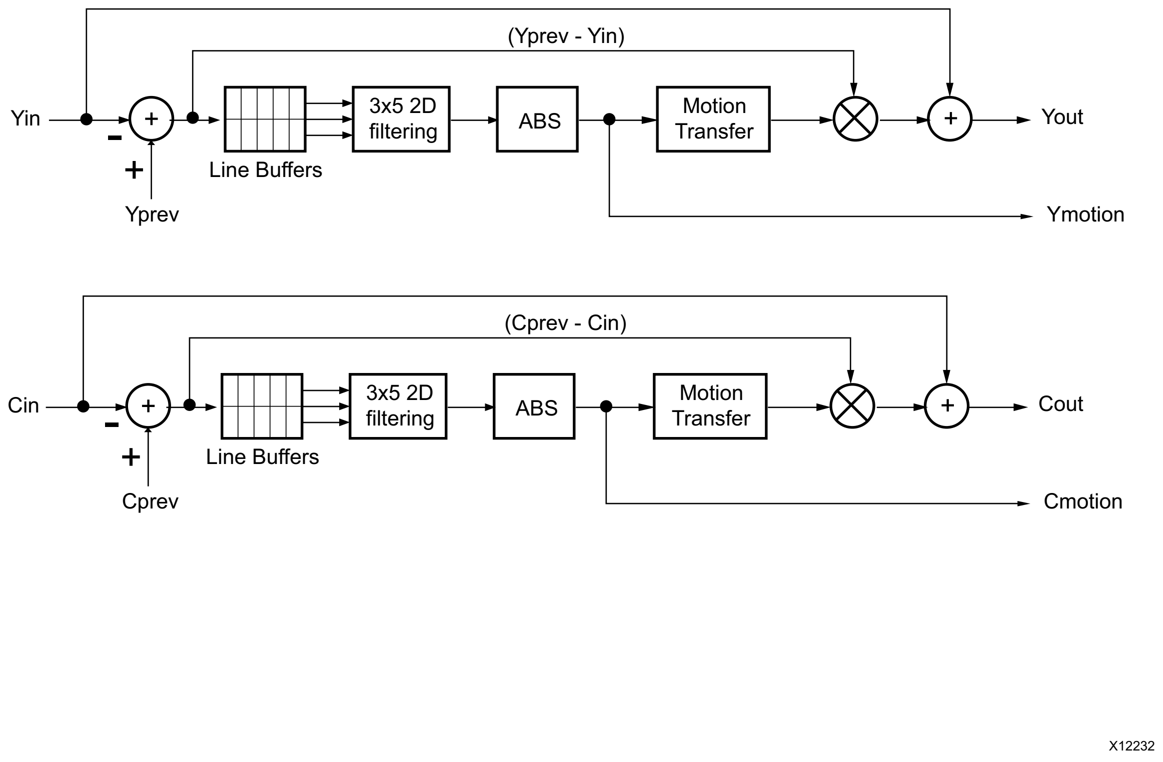


*Figure 3-1:* **Motion Adaptive Noise Reduction**

When the MANR core is used with an AXI4-Lite interface, the MTF function can be reprogrammed on a frame-by-frame basis using an arbitrary function. Refer to Motion Transfer Function (MTF) Access and Programming in Chapter 2 for more information. Best results have been demonstrated by using the monotonically decreasing portion of Gaussian or exponential functions. The MANR core is initialized from the tool 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 3-2. The Y-axis denotes the amount of recursion, and the X-axis denotes the amount of motion.

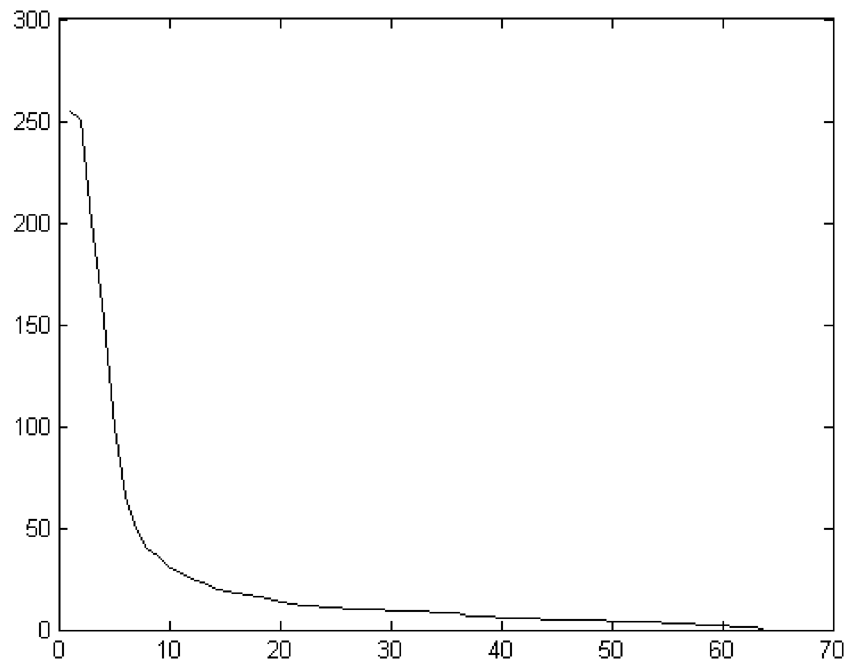


*Figure 3-2:* **Exponential MTF**

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. 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, hence incurs more recursion which results in more smoothing between the previous and current frames. Small changes in the pixel values 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 the function 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 3-2. This results in the same shape response, but varying degrees of recursion for the same shape. Shown in Figure 3-3 are example exponential MTFs with an attenuation of 0.75, 0.5, and 0.25 applied. The MANR core is generated with 8 initial MTFs preloaded into banks 0-7, with bank 0 containing the strongest, bank 7 containing the weakest setting. A strong MTF is well matched to a scenario with little motion and heavy noise load. A weak MTF is matched to a scenario with much motion and small noise load. 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 3-3, the curves illustrate how the attenuation factor is applied.

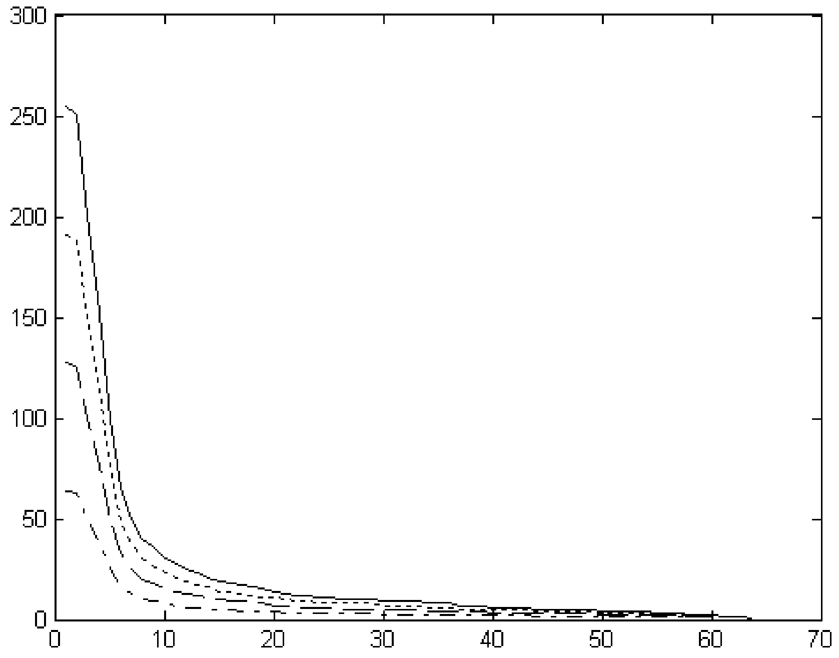


*Figure 3-3:* **MTF Settings**

The MANR core can store up to 16 MTF tables in memory. Only one table can be active in a given frame period. See MTF Storage and Switching in Chapter 3 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 16 separate MTF curves. Separate MTF curves are stored for Y and C channels.

Storing different MTFs can be useful in situations where the content being filtered differs in motion content. For example, a source might 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 exponential curve with a solid line shown in Figure 3-3 could be made active when the scene has little motion (because 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 on a frame-by-frame basis.

Xilinx provides eight pre-loaded MTF curves by default in the core. In addition, MTF values can be updated on a frame-by-frame basis, allowing a software application to easily control and optimize the MTF based on the expected source material and other conditions. To provide information about the content being processed, the MANR core measures estimated noise and motion content on a per-frame basis.

The application can periodically read out register values containing information on noise and motion, and select the active MTF bank that best matches the scenario, or download a custom MTF which is adapted to measurement data.

You can load custom MTFs into the remaining spaces in the block RAM, or overwrite the existing ones. Once overwritten, the default MTF is not available again unless the FPGA is reprogrammed. The MTF for each Y and C components consists of 64 discrete values that define the MTF curve. 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.

## 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 through external frame-buffer memory 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 to synchronize incoming data from these two interfaces.

## Output Interfaces

Video data is passed from the MANR core through three AXI4-Stream interfaces. Because the MANR operates as a recursive temporal filter, the output frame must be written into memory, where it is 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 interface. 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 optional `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` signal of the output AXI4-Stream interface. 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 optional `m_axis_motion` AXI4-Stream interface. Similar to video data, the interface handles 8-bit YC data, transmitted as the lowest 16-bits of the `TDATA` signal of the output AXI4-Stream interface. Luma motion occupies bits 7:0, and chroma motion occupies bits 15:8.

Figure 3-4 shows a typical use-case including the use of the AXI-VDMA block for external memory access.

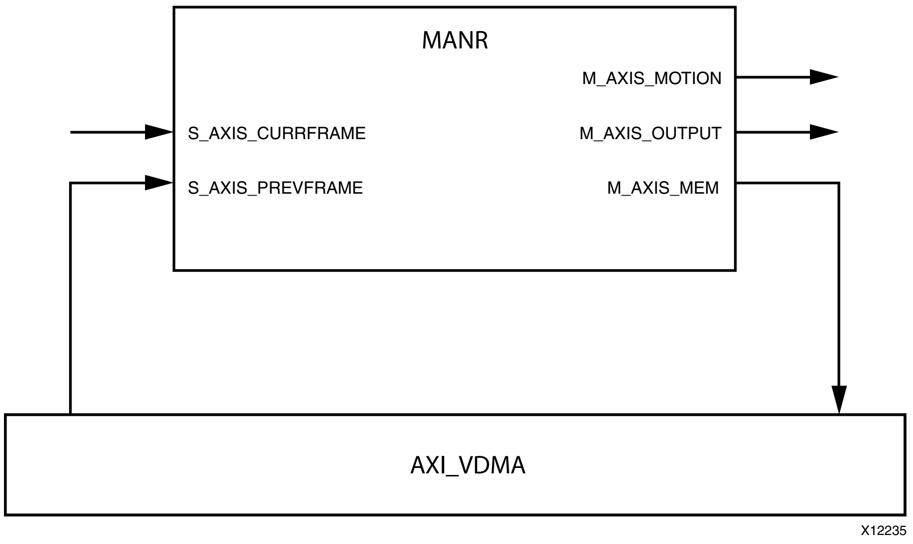


*Figure 3-4:* **Typical MANR Connectivity**

## Interrupts

The MANR core provides an internal interrupt controller with masking and enable features.

The core can generate a `FRAME_STARTED` interrupt, indicating that it has finished processing the previous frame and started working on a new frame. This signal can be useful for software to manage the core in the context of a larger pipeline.

## Use Models

Two examples are provided in this section that show the core usage for noise reduction only, and as the noise-reduction engine and motion-detection engine for a larger system.

Regardless of the application, the MANR core must have access to external memory using the AXI VDMA core. 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 3-5 and Figure 3-6, thick lines are used to indicate video data flow in the system.

### Use Model 1: Noise Reduction Application

Figure 3-5 shows an example where the MANR is used exclusively to reduce noise. In this case, streaming video data (current frames) is propagated to the MANR core directly, while previous frames are provided by the AXI-VDMA block.

Video data originates at a video source (for example, an HDMI™ technology source or a camera), with periodic timing signals, such as sync and blanking signals. The data might need to be pre-processed with Xilinx color-space-converter or chroma re-sampler cores to YCC422 format. Further processing can be undertaken before or after the MANR core.

The AXI-VDMA in Figure 3-5 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 timing controller shown in Figure 3-5 detect the video resolution, and make the detection results available for the system processor. The system processor distributes the resolution to the system components and programs, and initializes the frame-buffer mechanism in the AXI-VDMA.

The AXI4-Stream to Video Out module, after being configured, waits for the streaming video, then enables the Video Timing Controller (VTC) generator side to generate periodic video output timing signals.

When a system processor is not available, it is also possible to create a MANR sub-system to process video with pre-defined resolution. By configuring the constituent cores for no AXI4-Lite interface, the GUI allows you to specify one supported video resolution.

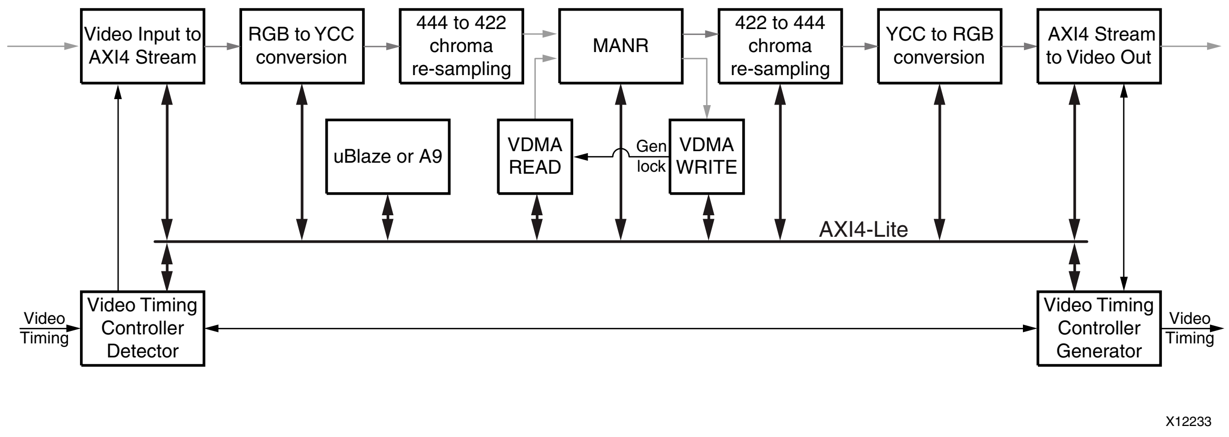

Video Input to AXI4 Stream
RGB to YCC conversion
444 to 422 chroma re-sampling
MANR
422 to 444 chroma re-sampling
YCC to RGB conversion
AXI4 Stream to Video Out
uBlaze or A9
VDMA READ
Gen lock
VDMA WRITE
AXI4-Lite
Video Timing
Video Timing Controller Detector
Video Timing Controller Generator
Video Timing
X12233

*Figure 3-5:* **Simple Noise Reduction**

## Use Model 2: Noise Reduction and Motion Detection

In Figure 3-6, the MANR is used to calculate and provide motion (change) data and noise reduction in a simple video processing system. The implicit 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 3-5, the MANR core noise-reduces the incoming video from a camera, or other period video source. However, in Figure 3-6, the MANR core also provides the video and motion data to a processing module through 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).

The MANR output streams can be passed to the Motion and Video Processing module(s) either directly, or indirectly through VDMA interfaces. When the VDMA output is written to memory, the `a_axis_output` stream may remain unused, because frame data has already been written to memory through the `mem_out` stream. Another pair of VDMA read ports can be used to read out motion and frame data from the frame buffer, isolating the input frame rate from the output frame rate.

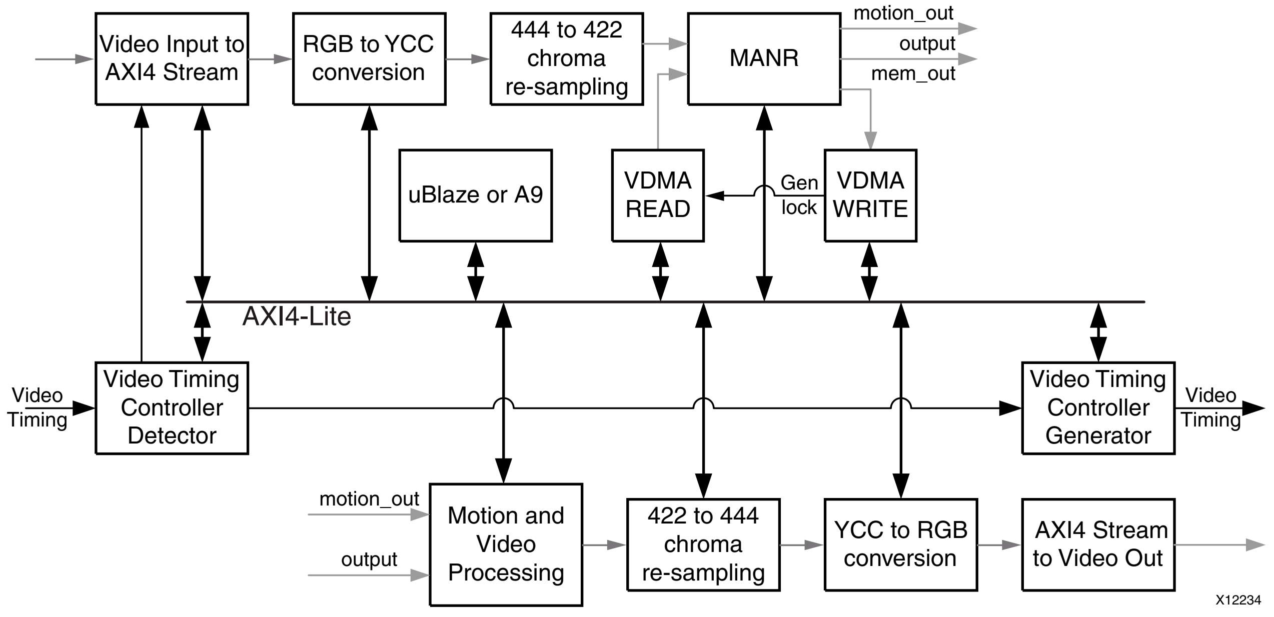


*Figure 3-6:* **Noise Reduction and Motion Processing**

## Clocking

The MANR core has one clock (`aclk`) that is used to clock the datapath of the entire core, and one optional clock, `s_axis_aclk`, which is used as the clock source for the optional AXI4-Lite interface.

## MANR Control and Timing

After reset, when using the AX4-Lite Control interface, you should initialize the registers to set up the frame size. Next, optional loading of an MTF can be done if the pre-loaded MTFs are not sufficient. Prior to enabling the MANR, ensure that the current and previous frame buffer locations are initialized with valid video data. This can be accomplished by enabling the core and enabling bypass mode in the control register. After a full frame has been committed to memory, as indicated by STATUS register bit 0, the bypass mode can be disabled and normal operation can begin. Figure 3-7 illustrates MANR initialization.

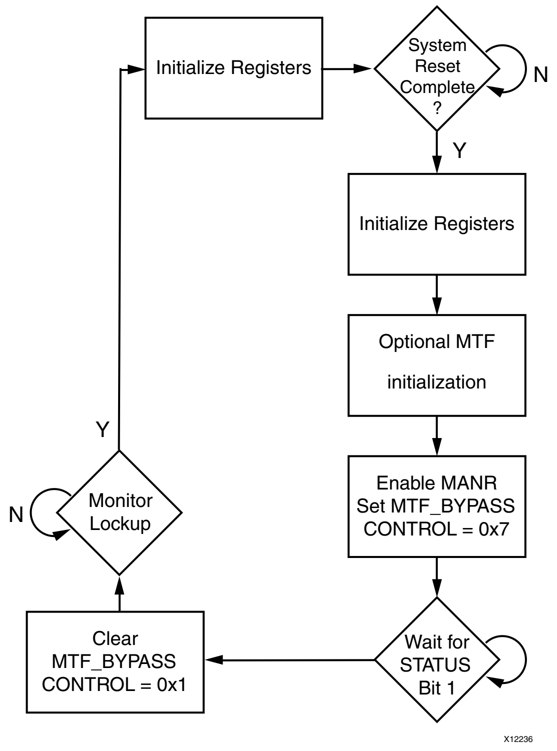


*Figure 3-7:* **MANR Initialization**

## Resets

The MANR core has one active-Low reset (`aresetn`) that resets the entire core. When using AXI4-Lite, an internal active-High software reset is combined with this signal.

## 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.

# Customizing and Generating the Core

This chapter includes information on the Vivado® Design Suite to customize and generate the core.

## Vivado Integrated Design Environment

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click on the selected IP or select the Customize IP command from the toolbar or popup menu.

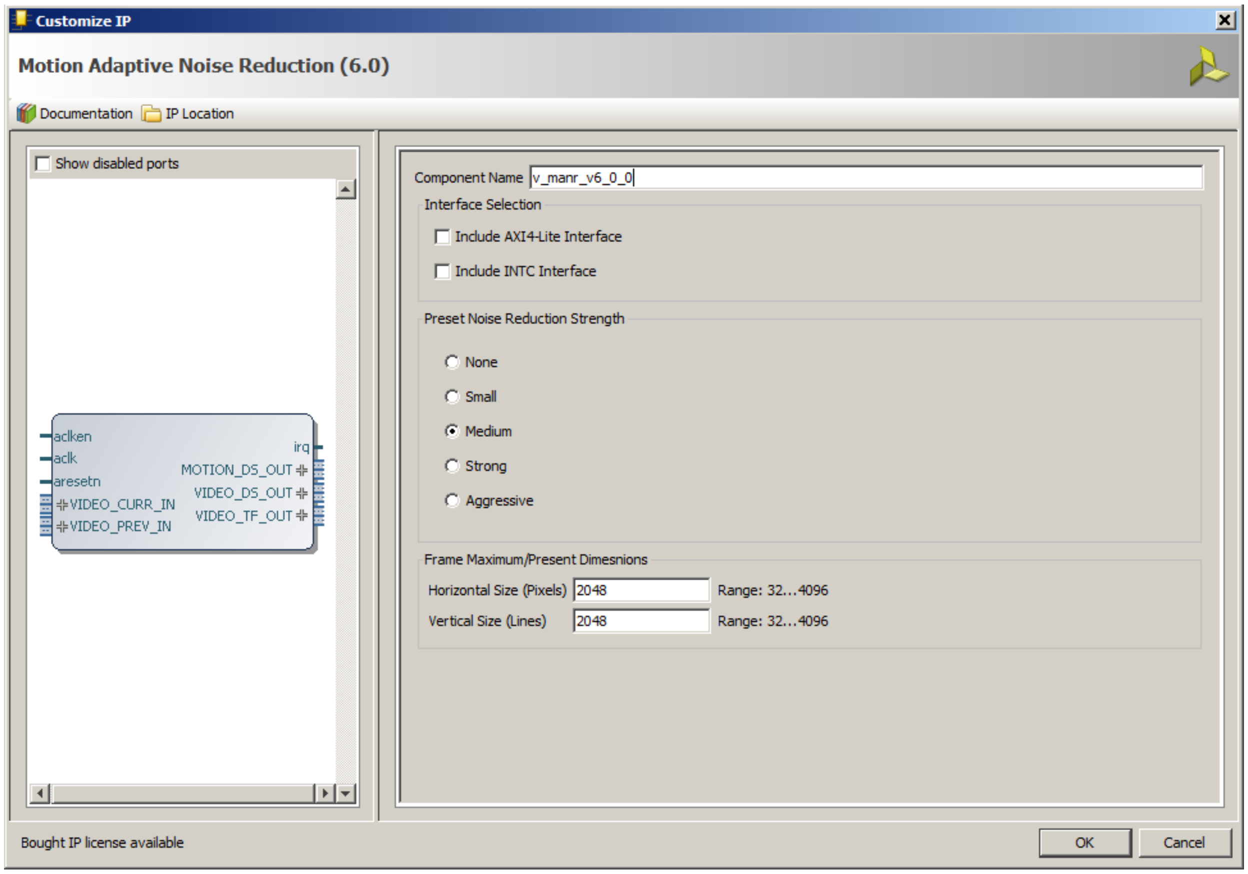
For details, see the sections, "Working with IP" and "Customizing IP for the Design" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8] and the "Working with the Vivado IDE" section in the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 10].

If you are customizing and generating the core in the Vivado IP Integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 11] for detailed information. IP Integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the `validate_bd_design` command in the tcl console.

***Note:*** Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version.

## Interface

The Xilinx® Motion Adaptive Noise Reduction (MANR) LogiCORE™ IP is easily configured to meet the developer's specific needs through the Vivado Design Suite Graphical User Interface (GUI). This section provides a quick reference to parameters that can be configured at generation time. Figure 4-1 shows the parameters for the MANR core in the Customize IP dialog box.



Motion Adaptive Noise Reduction (6.0)

☐ Show disabled ports

Component Name: v_manr_v6_0_0

Interface Selection
- ☐ Include AXI4-Lite Interface
- ☐ Include INTC Interface

Preset Noise Reduction Strength
- ○ None
- ○ Small
- ● Medium
- ○ Strong
- ○ Aggressive

Frame Maximum/Present Dimesnions

| | | |
|---|---|---|
| Horizontal Size (Pixels) | 2048 | Range: 32...4096 |
| Vertical Size (Lines) | 2048 | Range: 32...4096 |

Bought IP license available

*Figure 4-1:* **Customize IP Dialog Box**

The Customize IP dialog box 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_v6_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.

- **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.

## Output Generation

For details, see "Generating IP Output Products" in the *Vivado Design Suite User Guide: Designing with IP* (UG896).

# Constraining the Core

## Required Constraints

The only constraints required are clock frequency constraints for the video clock, `clk`, and the AXI4-Lite clock, `s_axi_aclk`. Paths between the two clock domains should be constrained with a `max_delay` constraint and use the `datapathonly` flag, causing setup and hold checks to be ignored for signals that cross clock domains. These constraints are provided in the XDC constraints file included with the core.

# C Model Reference

This chapter 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_v6_0 core
- Library module for the MANR core function
- Available for 32 and 64-bit Windows and 32 and 64-bit Linux platforms
- Supports all features of the HW core that affect numerical results
- Designed for rapid integration into a larger system model
- Example application C code is provided to show how to use the function

## Overview

The bit accurate C model for the LogiCORE™ IP MANR can be used on 32/64-bit Windows and 32/64-bit Linux platforms. The model is comprised of 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 41.

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.

## Unpacking and Model Contents

Unzip the `v_manr_v5_0_bitacc_model` file, containing the bit accurate models for the MANR IP Core. This creates the directory structure and files in Table 6-1.

*Table 6-1:* **Directory Structure and Files of MANR C Model**

| File Name | Contents |
|---|---|
| Makefile | Makefile for running gcc via make for 32-bit and 64-bit Linux platforms |
| v_manr_v6_0_bitacc_cmodel.h | Model header file |
| yuv_utils.h | Header file declaring the YUV image / video container type and support functions including .yuv file I/O |
| rgb_utils.h | Header file declaring the RGB image / video container type and support functions |
| bmp_utils.h | Header file declaring the bitmap (.bmp) image file I/O functions. |
| video_utils.h | Header file declaring the generalized image / video container type, I/O and support functions |
| video_fio.h | Header file declaring support functions for test bench stimulus file I/O |
| run_bitacc_cmodel.c | Example code calling the C model |
| run_bitacc_cmodel_config.c | Example code calling the C model – uses command line and config file arguments |
| run_bitacc_cmodel.sh | Bash shell script that compiles and runs the model. |
| files included in the lin64.zip file | Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms. |
| libIp_v_manr_v6_0_bitacc_cmodel.so | Model shared object library |
| run_bitacc_cmodel | 64-bit Windows fixed configuration executable |
| files included in the lin.zip file | Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms. |
| libIp_v_manr_v6_0_bitacc_cmodel.so | Model shared object library |
| run_bitacc_cmodel | 32-bit Windows fixed configuration executable |
| files included in the nt64.zip file | Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms. |
| libIp_v_manr_v6_0_bitacc_cmodel.dll | Precompiled library file for 64-bit Windows platforms compilation |
| libIp_v_manr_v6_0_bitacc_cmodel.lib | Precompiled library file for 64-bit Windows platforms compilation |
| run_bitacc_cmodel.exe | 64-bit Windows fixed configuration executable |

*Table 6-1:* **Directory Structure and Files of MANR C Model *(Cont'd)***

| File Name | Contents |
|---|---|
| files included in the nt.zip file | Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms. |
| libIp_v_manr_v6_0_bitacc_cmodel.dll | Precompiled library file for 32-bit Windows platforms compilation |
| libIp_v_manr_v6_0_bitacc_cmodel.lib | Precompiled library file for 32-bit Windows platforms compilation |
| 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 6-2.

*Table 6-2:* **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_v6_0_bitacc_simulate(
      struct xilinx_ip_v_manr_v6_0_generics *generics,
      struct xilinx_ip_v_manr_v6_0_inputs *inputs,
      struct xilinx_ip_v_manr_v6_0_outputs*  outputs).
```

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

```
struct  xilinx_ip_v_manr_v6_0_generics manr_generics;
struct  xilinx_ip_v_manr_v6_0_inputs  manr_inputs;
struct xilinx_ip_v_manr_v6_0_outputs* manr_outputs
```

The declaration of these structures are in the `v_manr_v6_0_bitacc_cmodel.h` file.

Before making the function call, complete these steps:

1. Populate the *generics* structure:

   **nr_strength** - Between 0 and 7. Describes the strength of the initial noise reduction filter: 0 = Strongest, 7=Weakest.

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 6-4. This core can only process YC422.
     - **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 mtf variable. The variable mimics the internal MTF storage of the core, and is declared as `extern int mtf[16][64];` in `v_manr_v6_0_bitacc_cmodel.h.`
4. 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 44 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 6-3:* **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 6-4. |
| Data | Set of five pointers to three dimensional arrays containing data for image planes. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col]. In the MANR C model case, only one frame is processed at any one time. Consequently, the '[frame]' index is always set to 0. |

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

*Table 6-4:* **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 YCbCr image/video data |
| FORMAT_C422[1] | 3 | 422 format YUV video, (U,V chrominance channels horizontally sub-sampled) |
| FORMAT_C420 | 3 | 420 format YUV video, ( U,V sub-sampled both horizontally and vertically) |
| FORMAT_MONO_M | 3 | Monochrome (luminance) video with motion |
| 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 6-4. 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 you 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 46. 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_v6_0_bitacc_cmodel -Wl,-rpath,.
```

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

```
gcc -m32 -x c++ ../run_bitacc_cmodel.c -o run_bitacc_cmodel -L.
-lIp_v_manr_v6_0_bitacc_cmodel -Wl,-rpath,.
```

To run either 32- or 64-bit executables:

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

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

### Windows (32/64-bits)

The `v_manr_v6_0_bitacc_cmodel.zip` file includes all the 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_v6_0_bitacc_cmodel.h`
- `libIp_v_manr_v6_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_v6_0_bitacc_cmodel.dll`
2. Execute as follows:

```
Usage: c_model  -y <YUV filename>
                           -h <H Size (pixels)>
                           -v <V Size (lines)>
                           -f <number of frames to be processed>
                           -n <Noise-reduction strength (0 - 4)>
```

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_v6_0_bitacc_cmodel
-Wl,-rpath,.

#################################################
# Run model.
# Usage:
# ./run_bitacc_model -y <input_file>.yuv -h <hsize> -v <vsize> -f <#frames> -n <NR_strength>

# NR_strength:
# 0 = None
# 1 = Weak
# 2 = Medium
# 3 = Strong
# 4 = Aggressive
# Example:
# ./run_bitacc_cmodel -y ../video_in.yuv -h 1280 -v 720 -f 10 -n 2
#################################################
./run_bitacc_cmodel -y ../video_in.yuv -h 1280 -v 720 -f 10 -n 2
```

You can customize this shell script.

# Simulation

This chapter contains information about simulating IP in the Vivado® Design Suite environment. For comprehensive information about Vivado simulation components, as well as information about using supported third party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 12].

# Synthesis and Implementation

For details about synthesis and implementation, see “Synthesizing IP” and “Implementing IP” in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

# Detailed Example Design

No example design is available for the Motion Adaptive Noise Reduction core at the time.

For a comprehensive listing of the latest 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

# Test Bench

This chapter contains information about the provided test bench in the Vivado® Design Suite environment.

## Demonstration Test Bench

A demonstration test bench is provided with the core which enables you to observe core behavior in a typical scenario. This test bench is generated together with the core in Vivado Design Suite. You are encouraged to make simple modifications to the configurations and observe the changes in the waveform.

### Directory and File Contents

The following files are expected to be generated in the in the demonstration test bench output directory:

- `axi4lite_mst.v`
- `axi4s_video_mst.v`
- `axi4s_video_slv.v`
- `ce_generator.v`
- `tb_<IP_instance_name>.v`

### Test Bench Structure

The top-level entity is **`tb_<IP_instance_name>`**.

It instantiates the following modules:

- `DUT`

  The <IP> 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 generates ramp data and initiates AXI4-Stream transactions to provide video stimuli for the core and can also be used to open stimuli files generated from the reference C models and convert them into corresponding AXI4-Stream transactions.

  To do this, edit `tb_<IP_instance_name>.v`:

  a. Add define macro for the stimuli file name and directory path
     `define STIMULI_FILE_NAME<path><filename>`.

  b. Comment-out/remove the following line:
     `MST.is_ramp_gen(`C_ACTIVE_ROWS, `C_ACTIVE_COLS, 2);`
     and replace with the following line:
     `MST.use_file(`STIMULI_FILE_NAME);`

  For information on how to generate stimuli files, see *Chapter 4, C Model Reference*.

- `axi4s_video_slv`

  The AXI4-Stream slave module, which acts as a passive slave to provide handshake signals for the AXI4-Stream transactions from the core output, can be used to open the data files generated from the reference C model and verify the output from the core.

  To do this, edit `tb_<IP_instance_name>.v`:

  a. Add define macro for the golden file name and directory path
     `define GOLDEN_FILE_NAME "<path><filename>"`.

  b. Comment out the following line:
     `SLV.is_passive;`
     and replace with the following line:
     `SLV.use_file(`GOLDEN_FILE_NAME);`

  For information on how to generate golden files, see *Chapter 4, C Model Reference*.

- `ce_gen`

  Programmable Clock Enable (`ACLKEN`) generator.

# 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 9, 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 to read registers back, validating the core Version register and proving that it has been implemented in the design. This has been run regularly to validate new core versions during development.

# Migrating and Upgrading

This appendix contains information about migrating from an ISE design to the Vivado Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading their IP core, important details (where applicable) about any port changes and other impact to user logic are included.

## Migrating to the Vivado Design Suite

For information about migration to Vivado Design Suite, see *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 7].

## Upgrading in Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

### Parameter Changes

There are no parameter changes.

### Port Changes

There are no port changes.

### Other Changes

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

From v5.01.a to v6.0 of the MANR core, the following changes took place:

- Two-dimensional spatial filtering added to improve noise / motion detection.
- MTF storage was extended to 16 banks.

- Default MTF count doubled to 8 default curves
- Debug frame, line, and pixel counters were added.
- Improved AXI4-Stream stability with the AXI-VDMA, built-in synchronization of slave (input) AXI4-Stream channels at SOF and EOL.
- Improved error detection and recovery for SOF / EOL framing errors.
- FRAME_SIZE register address made consistent with the rest of the Xilinx Image and Video Processing Cores.

# Debugging

This appendix includes details about resources available on the Xilinx® Support website and debugging tools.

## Finding Help on Xilinx.com

To help in the design and debug process when using the MANR core, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support WebCase.

### Documentation

This product guide is the main document associated with the MANR core. For the Video over AXI4-Stream specification, see *AXI4-Stream Video IP and System Design Guide (UG934)* [Ref 5]. These guides, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

### Solution Centers

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

### Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are listed below, and can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

**Answer Records for the MANR Core**

AR 54528

## Contacting 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.

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- A block diagram of the video system that explains the video source, destination and IP (custom and Xilinx) used.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

***Note:*** Access to WebCase is not available in all cases. Please login to the WebCase tool to see your specific support options.

## Debug Tools

There are many tools available to address MANR core design issues. It is important to know which tools are useful for debugging various situations.

### Vivado Lab Tools

Vivado® lab tools insert logic analyzer and virtual I/O cores directly into your design. Vivado lab tools allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature represents the functionality in the Vivado IDE that is used for logic debugging and validation of a design running in Xilinx devices in hardware.

The Vivado lab tools logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

### Reference Boards

Various Xilinx development boards support MANR. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series evaluation boards
    - KC705
    - KC724

### C Model Reference

See *Chapter 4, C Model Reference* in this guide for tips and instructions for using the provided C model files to debug your design.

## Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado lab tools are a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado lab tools for debugging the specific problems.

### General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `LOCKED` port.
- If your outputs go to 0, check your licensing.

### Core Bypass Option

The bypass option facilitates establishing a straight through connection between input (AXI4-Stream slave) and output (AXI4-Stream master) interfaces bypassing any processing functionality.

Flag `BYPASS` (bit 2 of the `CONTROL` register) can turn bypass on (1) or off. Within the IP this switch turns off noise reduction in the AXI4-Stream path.

In bypass mode the core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output.

Starting a system with all processing cores set to bypass, then by turning bypass off from the system input towards the system output allows verification of subsequent cores with known good stimuli.

### Throughput Monitors

Throughput monitors enable monitoring processing performance within the core. This information can be used to help debug frame-buffer bandwidth limitation issues, and if possible, allow video application software to balance memory pathways.

Often times video systems, with multiport access to a shared external memory, have different processing islands. For example, a pre-processing sub-system working in the input video clock domain may clean up, transform, and write a video stream, or multiple video streams to memory. The processing sub-system may read the frames out, process, scale, encode, then write frames back to the frame buffer, in a separate processing clock domain.

Finally, the output sub-system may format the data and read out frames locked to an external clock.

Typically, access to external memory using a multiport memory controller involves arbitration between competing streams. However, to maximize the throughput of the system, different memory ports may need different specific priorities. To fine tune the arbitration and dynamically balance frame rates, it is beneficial to have access to throughput information measured in different video datapaths.

The `SYSDEBUG0` (0x0014) (or Frame Throughput Monitor) indicates the number of frames processed since power-up or the last time the core was reset. The `SYSDEBUG1` (0x0018), or Line Throughput Monitor, register indicates the number of lines processed since power-up or the last time the core was reset. The `SYSDEBUG2` (0x001C), or Pixel Throughput Monitor, register indicates the number of pixels processed since power-up or the last time the core was reset.

Priorities of memory access points can be modified by the application software dynamically to equalize frame, or partial frame rates.

### Evaluation Core Timeout

The MANR hardware evaluation core times out after approximately eight hours of operation. The output is driven to zero. This results in a black screen for RGB color systems and in a dark-green screen for YUV color systems.

## Interface Debug

### AXI4-Lite Interfaces

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

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

| Symptom | Solution |
|---|---|
| Readback from the Version Register through the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive. | Are the `S_AXI_ACLK` and `ACLK` pins connected?<br>The `VERSION_REGISTER` readout issue may be indicative of the core not receiving the AXI4-Lite interface. |
| Readback from the Version Register through the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive. | Is the core enabled? Is `s_axi_aclken` connected to `vcc`? Verify that signal `ACLKEN` is connected to either `net_vcc` or to a designated clock enable signal. |
| Readback from the Version Register through the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive. | Is the core in reset?<br>`S_AXI_ARESETn` and `ARESETn` should be connected to `vcc` for the core not to be in reset. Verify that the `S_AXI_ARESETn` and `ARESETn` signals are connected to either `net_vcc` or to a designated reset signal. |
| Readback value for the `VERSION_REGISTER` is different from expected default values | The core and/or the driver in a legacy project has not been updated. Ensure that old core versions, implementation files, and implementation caches have been cleared. |

Assuming the AXI4-Lite interface works, the second step is to bring up the AXI4-Stream interfaces.

## AXI4-Stream Interfaces

Table C-2 describes how to troubleshoot the AXI4-Stream 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 Vivado Lab Tools, 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 Vivado Lab Tools, measure the number of active AXI4-Stream transactions between `EOL` pulses. |

*Table C-2:* **Troubleshooting AXI4-Stream Interface *(Cont'd)***

| Symptom | Solution |
|---|---|
| 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 Vivado Lab Tools, 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 core keeps its s_axis_video_tready input low. Afterwards, the core should assert s_axis_video_tready automatically.<br>Is m_axis_video_tready low? If so, the core cannot send data downstream, and the internal FIFOs are full. |
| m_axis_video_tvalid stuck low, the downstream core is not receiving data | • No data is generated during the first two lines of processing.<br>• 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. |
| Data samples lost between Upstream core and this core. Inconsistent EOL and/ or SOF periods received. | • Are the Master and Slave AXI4-Stream interfaces in the same clock domain?<br>• Is proper clock-domain crossing logic instantiated between the upstream core and this core (Asynchronous FIFO)?<br>• Did the design meet timing?<br>• Is the frequency of the clock source driving the ACLK pin lower than the reported Fmax reached? |
| Data samples lost between Downstream core and this core. Inconsistent EOL and/ or SOF periods received. | • Are the Master and Slave AXI4-Stream interfaces in the same clock domain?<br>• Is proper clock-domain crossing logic instantiated between the upstream core and this core (Asynchronous FIFO)?<br>• Did the design meet timing?<br>• Is the frequency of the clock source driving the ACLK pin lower than the reported Fmax reached? |

If the AXI4-Stream communication is healthy, but the data seems corrupted, the next step is to find the correct configuration for this core.

## Other Interfaces

Table C-3 describes how to troubleshoot third-party interfaces.

*Table C-3:* **Troubleshooting Third-Party Interfaces**

| Symptom | Solution |
|---|---|
| Severe color distortion or color-swap when interfacing to third-party video IP. | Verify that the color component logical addressing on the AXI4-Stream `TDATA` signal is in according to *Data Interface* in Chapter 2. If misaligned:<br>In HDL, break up the `TDATA` vector to constituent components and manually connect the slave and master interface sides. |
| Severe color distortion or color-swap when processing video written to external memory using the AXI-VDMA core. | Unless the particular software driver was developed with the AXI4-Stream TDATA signal color component assignments described in *Data Interface* in Chapter 2 in mind, there are no guarantees that the software correctly identifies bits corresponding to color components.<br>Verify that the color component logical addressing `TDATA` is in alignment with the data format expected by the software drivers reading/writing external memory. If misaligned:<br>In HDL, break up the `TDATA` vector to constituent components, and manually connect the slave and master interface sides. |

# Application Software Development

This appendix details the use of the software drivers that are included with SDK.

## Device Drivers

### Driver Files

The MANR core is delivered with a software driver written in the C programming language that you can use to control the core using a system processor. 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.

### API Functions

This section describes the functions included for the driver generated for the MANR core. To use the API functions provided, the following header files must be included in your 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.

#### 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.

# Additional 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:

1. AMBA® AXI4-Stream Protocol Specification
2. *LogiCORE IP FIFO Generator Product Guide* (PG057)
3. *Xilinx AXI Reference Guide* (UG761)
4. *LogiCORE IP AXI Interconnect Product Guide (*PG059*)*
5. *AXI4-Stream Video IP and System Design Guide* (UG934)
6. *AXI Reference Guide* (UG761)
7. *ISE to Vivado Design Suite Migration Guide* (UG911)
8. *Vivado Design Suite User Guide: Designing with IP* (UG896)
9. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
10. *Vivado Design Suite User Guide: Getting Started* (UG910)
11. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
12. *Vivado Design Suite User Guide: Logic Simulation (*UG900*)*

## 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. |
| 07/25/2012 | 3.0 | Updated core to v5.00.a. Added support for Vivado Design Suite implementations. |
| 12/18/2012 | 3.1 | • Updated for core v5.01.a, ISE Design Suite v14.4 and Vivado Design Suite v2012.4. • Updated resource and performance data. • Updated register interface documentation. • Added Appendix C, Debugging. |
| 03/20/2013 | 4.0 | • Updated for core v6.0 • Removed ISE • The following sections: Throughput and Register Space. |
| 10/02/2013 | 6.0 | Synch document version with core version. Updated Constraints, C-Model contents, and Hardware Debug. |

## Notice of Disclaimer