LogiCORE™ IP Motion Adaptive Noise Reduction v2.0 Bit Accurate C Model

User Guide

UG826 May 6, 2011

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/06/11	1.0	Initial Xilinx release.

Table of Contents

About This Guide

This user guide provides information about the Xilinx® LogiCORE™ IP Video Motion Adaptive Noise Reduction (MANR) v2.0 bit accurate C model core.

Guide Contents

This manual contains these chapters:

- Chapter 1, Introduction introduces the bit accurate C model for the Xilinx LogiCORE IP MANR v2.0 core, which has been developed primarily for system level modeling.
- Chapter 2, User Instructions provides information on the C model directory structure, files, installation, and software requirements.
- Chapter 3, Interface provides information on the C model interface, including defining the inputs, generics and output of the MANR core.
- Chapter 4, C Model Example Code provides an example C file along with the Win32 version of the executable for this example.

Additional Resources

To find additional documentation, see the Xilinx website at:

www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

www.xilinx.com/support/mysupport.htm.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**`lowpwr =`**{**`on`**|**`off`**}
Vertical bar |	Separates items in a list of choices	**`lowpwr =`**{**`on`**|**`off`**}
Angle brackets < >	User-defined variable or in code samples	<directory name>

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Convention	Meaning or Use	Example
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	**usr_teof_n** is active low.

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Introduction

This document introduces the bit accurate C model for the Xilinx® LogiCORE™ IP Motion Adaptive Noise Reduction v2.0 core, which has been developed primarily for system level modeling.

Features

- Bit accurate with v_manr_v2_0 core
- Library module for the MANR core function
- Available for 32-bit Windows 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 Xilinx® LogiCORE IP MANR v2.0 can be used on 32-bit Windows and 64-bit Linux platforms. The model comprises a set of C functions, which reside in a statically linked library (shared library). Full details of the interface to these functions are provided in Chapter 3, Interface.

The main features of the C model package are:

- **Bit Accurate C Model** - produces the same output data as the MANR v2.0 core on a frame-by-frame basis. However, the model is not cycle accurate, as it does not model the core's latency or its interface signals.
- **Application Source Code** - uses the model library function. This can be used as example code showing how to use the library function. However, it also serves these purposes:
  - **Input .yuv file** is processed by the application; 8-bit YUV422 format accepted.
  - **Output .yuv file** is generated by the application; 8-bit YUV422 format generated.
  - **Report.txt file** is generated for run time status and error messages.

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

Additional Core Resources

For detailed information and updates about the MANR v2.0 core, see these documents located on the core product page at: http://www.xilinx.com/products/ipcenter/EF-DI-IMG-MA-NOISE.htm

- *Motion Adaptive Noise Reduction v2.0 Data Sheet (DS841)*
- *Motion Adaptive Noise Reduction v2.0 Release Notes*

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the MANR v2.0 core.

Xilinx provides technical support for use of this product as described in this user guide (*LogiCORE Motion Adaptive Noise Reduction Bit Accurate C Model User Guide*).

Xilinx cannot guarantee functionality or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the MANR v2.0 core and the accompanying documentation.

MANR v2.0 Bit Accurate C Model and IP Core

For comments or suggestions about the MANR v2.0 core and bit accurate C model, submit a WebCase from http://www.xilinx.com/support/clearexpress/websupport.htm. Be sure to include the following information:

- Product name
- Core version number
- Explanation of your comments

Document

For comments or suggestions about the documentation for the MANR v2.0 core and bit accurate C model, submit a WebCase from http://www.xilinx.com/support/clearexpress/websupport.htm. Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

XILINX®

Chapter 2

User Instructions

Unpacking and Model Contents

Unzip the `v_manr_v2_0_bitacc_model.zip` file, containing the bit accurate models for the MANR IP Core. This creates the directory structure and files in Table 2-1.

Table 2-1: **Directory Structure and Files of the MANR v2.0 Bit Accurate C Model**

File Name	Contents
README.txt	Release notes.
ug826_v_manr.pdf	LogiCORE IP Motion Adaptive Noise Reduction Bit Accurate C Model User Guide.
v_manr_v2_0_bitacc_cmodel.h	Model header file.
rgb_utils.h	Header file declaring the RGB image/video container type and support functions.
yuv_utils.h	Header file declaring the YUV (.yuv) image file I/O functions.
video_utils.h	Header file declaring the generalized image/video container type, I/O and support functions.
run_bitacc_cmodel.c	Example code calling the C model.
video_in.yuv	10 frame video sequence for test purposes. Video format: 1280x720; YC422; 8 bits.
video_in.hdr	YUV file header.
/lin64	Lin64 library directory.
libIp_v_manr_v2_0_bitacc_cmodel.so	Precompiled v_manr_v2_0 dynamically linked shared object file for lin64 compilation.
libstlport.so.5.1	STL library, referenced by libIp_ v_manr_v2_0_bitacc_cmodel.so (Linux platforms only).
run_bitacc_cmodel.sh	Bash shell script that compiles and runs the model.
/win32	Win32 library directory.
libIp_v_manr_v2_0_bitacc_cmodel.lib	Precompiled statically linked MANR library file for win32 compilation.

Software Requirements

The MANR v2.0 C models were compiled and tested with the software listed in Table 2-2.

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

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

Chapter 3

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_v2_0_bitacc_simulate(
      struct xilinx_ip_v_manr_v2_0_generics* generics,
      struct xilinx_ip_v_manr_v2_0_inputs*   inputs,
      struct xilinx_ip_v_manr_v2_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_v2_0_generics manr_generics;
struct  xilinx_ip_v_manr_v2_0_inputs  manr_inputs;
struct  xilinx_ip_v_manr_v2_0_outputs  manr_outputs
```

The declaration of these structures are in the `v_manr_v2_0_bitacc_cmodel.h` file.

Before making the function call, complete these steps:

1. Populate the *generics* structure:

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

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

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

   - **video_in** - Video structure that comprises these elements:
     - **bits_per_component** - Must be set to 8.
     - **cols** - Horizontal image size: 32 to 1920.
     - **rows** - Vertical image size: 32 to 1080.
     - **frames** - Set to 1; this function processes *one frame at a time.*
     - **mode** - Defines the chroma format (RGB, YUV422, and so on); see Table 3-2. This core can only process YC422 or YC420.
     - **data** - This is the frame of video data to be processed, arranged in raster form.
   - **mtf** - MTF Look-up table. This is a 1D array of 64 integers in the range 0 to 255, which represents the Motion Transfer Function. For information on the values that must populate this array, see the *LogiCORE IP Motion Adaptive Noise Reduction v2.0 Data Sheet* (DS841).

3. Populate the *outputs* structure.
   - **video_out** - Video structure that comprises the same elements as the video_in structure element described previously.
   - **yc_in_fid** - not used by the C model core; used in the wrapper. Default: set string to "".

- **ycm_out_fid** - not used by the C model core; used in the wrapper. Default: set string to "".

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 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 v2.0 reference model using the general purpose `video_struct` structure, defined in `video_utils.h`:

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

Table 3-1: **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*
Cols	Number of columns per frame*
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 3-2.
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.

*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 3-2: **Named Constants for Video Modes With Corresponding Planes and Representations**

Mode	Planes	Video Representation
FORMAT_MONO	1	Monochrome – luminance only
FORMAT_RGB	3	RGB image/video data
FORMAT_C444	3	444 YUV, or YCrCb image/video data
FORMAT_C422	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

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 3-2. 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);
```

XILINX®

Chapter 4

C Model Example Code

An example C file, `run_bitacc_cmodel.c`, is provided along with the Win32 version of the executable for this example (`run_bitacc_cmodel.exe`). This C file has these characteristics:

- Contains an example of how to write an application that makes a function call to the MANR C model core function.
- Contains an example of how to populate the video structures at the input and output, including allocation of memory to these structures.
- Uses a YUV file reading function to extract video information for use by the model.
- Uses a YUV file writing function to provide an optional output YUV file, which allows the user to visualize the result of the MANR operation.

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

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

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

For more information on the specifics of the MANR operation, see the *LogiCORE IP Motion Adaptive Noise Reduction v2.0 Data Sheet* (DS841).

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. The `yuv_util.h` and `video_util.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, such as YUVPlayer.
  - 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, such as YUVPlayer.

Compiling the MANR v2.0 C Model With Example Wrapper

Linux (64-bit)

To compile the example code, perform these steps:

1. Set your $LD_LIBRARY_PATH environment variable to include the root directory where you unzipped the model zip file, as shown in this example:

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

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

   `libstlport.so.5.1`

   `libIp_v_manr_v2_0_bitacc_cmodel.so`

3. In the root directory, compile with the GNU C Compiler using this command:

```
gcc -x c++ run_bitacc_cmodel.c -o run_bitacc_cmodel -L. -
lIp_v_ycrcb2rgb_v3_0_bitacc_cmodel -lIp_v_manr_v2_0_bitacc_cmodel -Wl,-rpath,.
```

Windows (32-bit)

Precompiled `library v_manr_v2_0_bitacc_cmodel.lib`, and top-level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. The following is an example using Microsoft Visual Studio. In Visual Studio create a new, empty Win32 Console Application project. To existing items, add:

- **libIp_v_manr_v2_0_bitacc_cmodel.lib** to the "Resource Files" folder of the project
- **run_bitacc_cmodel.c** to the "Source Files" folder of the project
- **v_manr_v2_0_bitacc_cmodel.h** to "Header Files" folder of the project
- **yuv_utils.h** to the "Header Files" folder of the project
- **rgb_utils.h** to the "Header Files" folder of the project
- **video_utils.h** to the "Header Files" folder of the project

After the project is created and populated, it must be compiled and linked (built) to create a Win32 executable. To perform the build step, select **Build Solution** from the Build menu. An executable matching the project name is created either in the Debug or Release subdirectories under the project location based on whether "Debug" or "Release" was selected in the "Configuration Manager" in the Build menu.

Running the Generated Executables

Usage:

```
run_bitacc_cmodel in_yuv_file out_yuv_file #frames #cols #rows chroma_mode, nr_strength
```

in_yuv_file: Path/name of the input file - must be .yuv.

out_yuv_file: Path/name of the output file - must be .yuv.

frames: Number of frames in input file.

cols: Number of columns in image.

rows: Number of rows in image.

chroma_mode: chroma sub-sampling:

1: 420

2: 422

nr_strength: Noise reduction strength range 0-4:

0: None

1: Weak

2: Medium

3: Strong

4: Aggressive

Example:

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

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 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_v2_0_bitacc_cmodel -
Wl,-rpath,.


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

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

The user can customize this shell script.