H.264 CABAC Encoder Core v1.0

User Guide

UG435 (v1.1) May 29, 2007

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/15/07	1.0	Initial Xilinx release.
05/29/07	1.1	Revised “Manual Installation” in Chapter 2.

Contents

XILINX®

Preface

About This Guide

This document is intended to guide the user through all aspects of installation, demonstration, simulation, verification, and general usage of the H.264 CABAC Encoder core. It should be read in conjunction with the Xilinx H.264 CABAC Encoder Core Product Specification (DS603).

This manual contains the following chapters:

- Chapter 1, "Introduction"
- Chapter 2, "Installing the H. 264 CABAC Encoder Core"
- Chapter 3, "Designing with the H.264 CABAC Encoder Core"
- Chapter 4, "Simulating the H.264 CABAC Encoder Core"
- Chapter 5, "System Verification"
- Appendix A, "Additional Information"

Additional Resources

To find additional documentation, see the Xilinx website at:

http://www.xilinx.com/literature.

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:

http://www.xilinx.com/support.

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*

Convention	Meaning or Use	Example
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 *Development System Reference 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.
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**}
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;*

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.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-II Platform FPGA User Guide.*
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Introduction

This user guide is required reading for an engineer using or considering using the Xilinx H.264 CABAC Encoder core. This product is released in netlist form through ZIP files. The document is intended to guide the user through all aspects of installation, demonstration, simulation, verification, and general usage of the H.264 CABAC Encoder core. It should be read in conjunction with the Xilinx H.264 CABAC Encoder Core Product Specification (DS603).

About the Core

The Xilinx H.264 CABAC Encoder core performs entropy coding of macroblock, motion vector, and other data generated within the H.264 standard (see [Ref 1]).

As deliverables, Xilinx provides a data structure input interface for slice and macroblock related data which generates the resulting entropy codec bitstream at the output. Additional output of the length of bytes is generated to assist in rate control on a macroblock or slices basis.

The internal core operates through FIFO-like interfaces and produces bitstreams up to 80 Mega bits per second (Mbps).

Recommended Design Experience

Although the H.264 CABAC Encoder core is a fully-verified solution, the challenge associated with implementing a complete design varies, depending on the configuration and functionality of the application. For best results, previous experience building high performance, pipelined FPGA designs using Xilinx implementation software and system-level user constraints file (UCF) is recommended.

Contact your local Xilinx representative for a closer review and estimation for your specific site requirements.

Additional Core Resources

For detailed information and updates about the H.264 CABAC Encoder core, see the documents, located on the H.264 CABAC Encoder product page. In general, this document should always be used in conjunction with the following:

- The Xilinx *H.264 CABAC Encoder Core v1.0 Data Sheet*, (DS603).
- The MPEG4 Part 10 Specification [Ref 1]
- JM10.1 H.264 Codec Reference C Code

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team of engineers who have expertise using the H.264 CABAC Encoder core.

Xilinx will provide technical support for this product as described in this user guide.

Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow the guidelines in this document.

Feedback

Xilinx welcomes comments and suggestions about the H.264 CABAC Encoder core and the accompanying documentation. For comments or suggestions about the H.264 CABAC Encoder core, submit a WebCase from www.xilinx.com/support. Be sure to include the following information:

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

For comments or suggestions about this document, submit a WebCase from www.xilinx.com/support. Be sure to include the following information:

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

References

1. ITU-T/ISO/IEC, *Advanced Video Coding for Generic Audiovisual Services*, H.264 03/2005.

Chapter 2

Installing the H. 264 CABAC Encoder Core

This chapter provides instructions for installing the H.264 CABAC Encoder core.

The H.264 CABAC fixed netlist is provided under the Xilinx LogiCORE™ Site License Agreement, which conforms to the terms of the SignOnce IP License standard defined by the Common License Consortium.

The user installs the core by performing a manual installation after downloading the core from the Web.

System Requirements

- Windows
  - Windows® 2000 Professional with Service Pack 2 or greater
  - Windows XP Professional Service Pack 2 or greater
- Software
  - Xilinx ISE™ 9.1i with Service Pack 1
  - ModelSim® 6.1c SE
  - ActivePerl 5.8.3
- Hardware
  - Annapolis Micro Systems WildCard™ 4 Platform
  - API 2.6
  - Driver 3.3
  - Firmware 1.6

Manual Installation

Full System Hardware Evaluation Core

The user can obtain a Full System Hardware Evaluation version of the core at no cost by clicking the **Evaluate** link on the core's product page:

www.xilinx.com/h264/cabac/index.htm

The user can then download the evaluation ZIP file from the product-specific evaluation page.

This version of the core lets the user fully integrate the core into an FPGA design, place and route the design, evaluate timing, and perform back-annotated gate-level simulation of the core using the demonstration test bench provided.

In addition, with the evaluation core, the user can generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device 8 hours before *timing out* (ceasing to function) at which time it can be reactivated by reconfiguring the device.

Full Core

To access the full-featured version of the core:

1. License the core through your local Xilinx sales office.
2. Xilinx will send a Product Serial Number by mail.
3. To register for Lounge Access:
   a. If you are new to Xilinx, click **Create an Account** and follow the instructions.
   b. Follow the lounge registration instructions on the **Order & Register** link located on the main product information page: www.xilinx.com/h264/cabac/index.htm
   c. Xilinx Development Systems Customer Service will review your access request and typically grant access to the lounge in 48 hours. (Contact them directly by phone if you need faster turnaround.)
   d. After you have been granted access by Customer Service, go back to the main product page and click the **Access Lounge** link to log in to the lounge.
   e. Download the design files listed (in step 4) from the product lounge, saving them to a suitable directory, for example, `CABAC_ver1_0`.
4. Three zip files are provided. Unzip them all into the root directory. They are as follows (x_y is the release version number, for example, 1_0):
   a. `H.264_CABAC_releasex_y_utils.zip` - Unzip this first.
   b. `H.264_CABAC_releasex_y_release_netlists.zip`
   c. `H.264_CABAC_releasex_y_input_sequences.zip`
5. Allow the extractor utility you use to overwrite all existing files and maintain the directory structure defined in the archive. See Appendix A, "Additional Information."

Netlists

The netlists provided in this release of the H.264 CABAC Encoder core are all in the `\ReleasedNetlists` directory. These netlists have been synthesized using Synplify_Pro 8.6.2 for Spartan™-3E (S3), Virtex™-4 (V4), and Virtex-5 (V5) devices. Four Netlist groups are generated for each device speed grade and device family:

```
cabac_top_S3-4_CIF.edf
cabac_top_S3-5_CIF.edf
cabac_top_S3-4_720.edf
cabac_top_S3-5_720.edf
cabac_top_S3-4_601.edf
cabac_top_S3-5_601.edf
cabac_top_S3-4_1080.edf
cabac_top_S3-5_1080.edf
cabac_top_V4-10_CIF.edf
cabac_top_V4-11_CIF.edf
cabac_top_V4-12_CIF.edf
cabac_top_V4-10_720.edf
cabac_top_V4-11_720.edf
cabac_top_V4-12_720.edf
cabac_top_V4-10_601.edf
cabac_top_V4-11_601.edf
cabac_top_V4-12_601.edf
cabac_top_V4-10_1080.edf
cabac_top_V4-11_1080.edf
cabac_top_V4-12_1080.edf
cabac_top_V5-1_CIF.edf
cabac_top_V5-2_CIF.edf
cabac_top_V5-3_CIF.edf
cabac_top_V5-1_720.edf
cabac_top_V5-2_720.edf
cabac_top_V5-3_720.edf
cabac_top_V5-1_601.edf
cabac_top_V5-2_601.edf
cabac_top_V5-3_601.edf
cabac_top_V5-1_1080.edf
cabac_top_V5-2_1080.edf
cabac_top_V5-3_1080.edf
```

To learn how to instantiate the cores in your system, refer to the section **Chapter 3, "Designing with the H.264 CABAC Encoder Core."**

Chapter 3

Designing with the H.264 CABAC Encoder Core

This chapter describes how to include an H.264 CABAC Encoder core into the next hierarchy of system architecture.

Source Files Release

To help the user design the core into his system, the release provides sample source files.

Note: **These source files are for reference only**. Neither the core source code nor the HDL libraries have been provided. The user should not attempt to use these files directly for synthesis or simulation:

1. Wrapper code file:

   `\HDL\CABAC\src\CABAC_ top_main_wrap.vhd (CABAC_top_main instantiated here)`
   `\HDL\CABAC\src\cabac_TOP_wrap.vhd (CABAC_top_main instantiated here with FIFOs)`

2. MTI test bench-specific code:

   `\HDL\CABAC\src\CABAC_top_core_architecture.vhd (CABAC_top` instantiated here)
   `\HDL\CABAC\src\CABAC_top_core_entity.vhd (entity for above)`
   `\HDL\CABAC\src\CABAC_top.vhd` (IF1 wrapper – instantiates `CABAC_top_core` and includes FIFOs)
   `\HDL\CABAC\Testbench\CABAC_top_TB.vhd`

3. XLIB (FIFOs)

   `\HDL\CABAC\src\XLIB\src\MemXLib_arch.vhd`
   `\HDL\CABAC\src\XLIB\src\MemXLib_util.vhd`

4. WildCard code:

   `\Platform\Wildcard\HW\HDL\virtual_socket\vs_fifo_move_CABAC.vhd`
   (instantiates `cabac_top` and includes FIFOs)

CABAC in the H.264 Encoder

Figure 3-1 shows CABAC in a typical implementation of a CABAC encoder. For more information on the system-level integration, see the H.264 specification [Ref 1].

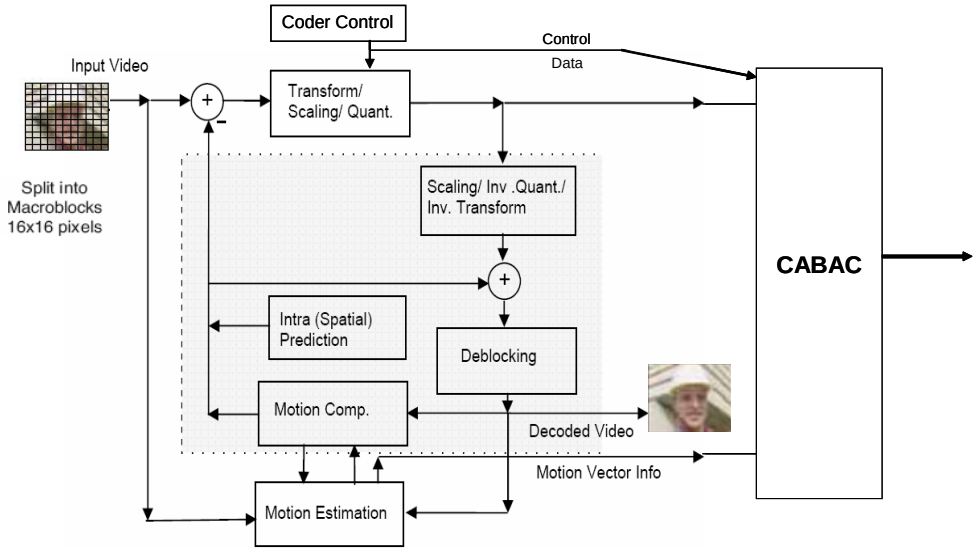

Figure 3-1: **H.264 Encoder**

CABAC Encoder Use Model

A typical usage model of the CABAC encoder is shown in Figure 3-2. Slice and macroblock related information are sent by the user through two separate input FIFOs added by the user. An output FIFO is used to collect the encoded data stream. Additionally, with the stream length of the current slice given by the signal **`Ecodestrm_len[29:0]`** and the end of the current macroblock signal **`Ecodestrm_MB_done`** signal, the user has the ability to count the bit length of the current macroblock. This is useful for any output rate flow control scheme.

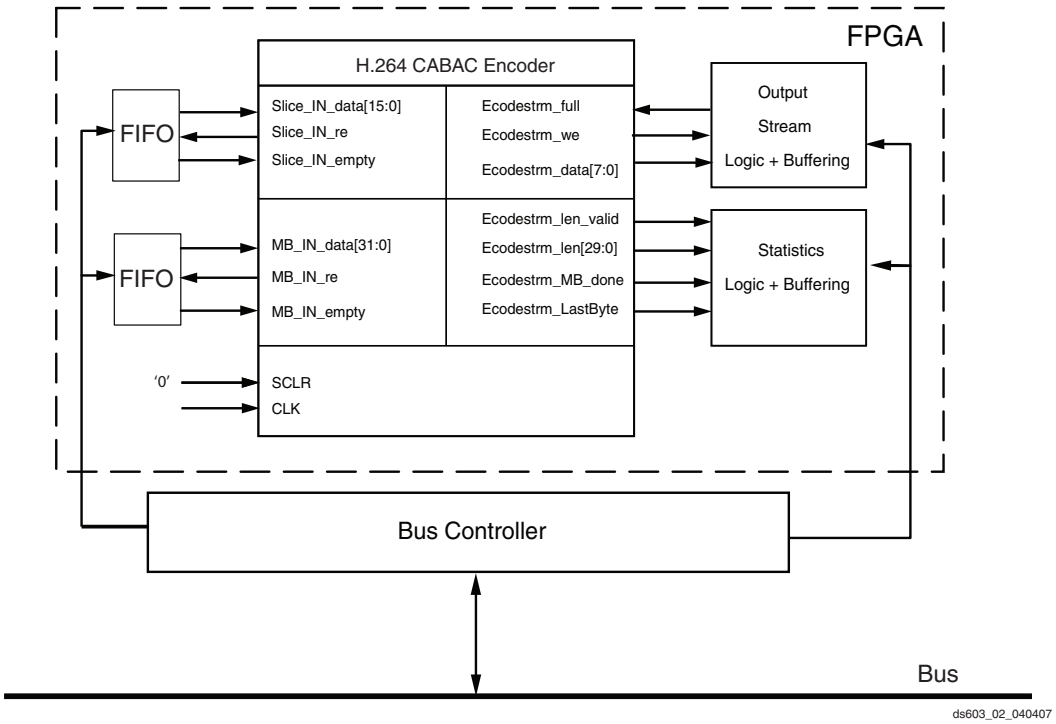

Figure 3-2: **Typical CABAC Usage**

Chapter 4

Simulating the H.264 CABAC Encoder Core

This chapter describes a test bench architecture that was created in the ModelSim environment for simulation of the CABAC cores. It does *not* describe the verification platform. The simulation described may be used for visualization of the I/O signals at the periphery of the cores. Although the same precompiled libraries are used in the verification of CABAC, this process is described in detail in "Chapter 5, "System Verification."

Test Bench Release

For running, viewing, and understanding the test bench and the architectures used for CABAC, the release provides the following files under **\HDL\CABAC**:

1. Libraries of object (precompiled) source:

   ```
   \Testbench\cabac_top\work\
   \Testbench\cabac_top\memxlib\
   \Testbench\cabac_top\mpeg4xlib\
   \Testbench\cabac_top\Sim_tools\
   ```

2. Test bench stimulus files:

   ```
   \Testbench\cabac_top\stimuli\SliceObj_in.txt (input stimulus)
   \Testbench\cabac_top\stimuli\MBObjA_in.txt (input stimulus)
   \Testbench\cabac_top\stimuli\MBObjB_in.txt (input stimulus)
   ```

3. ModelSim-specific script files:

   ```
   \Testbench\cabac_top\cabac_TOP_spv2_prim.do
   \Testbench\cabac_top\cabac_TOP_spv2_core.do
   \Testbench\cabac_top\cabac_TOP_spv2_user.do
   \Testbench\cabac_top\vsim_gui.bat
   ```

4. Test bench source file:

   ```
   \Testbench\cabac_top\cabac_TOP_spv2_TB.vhd
   ```

Running the Test Bench

To run the test bench, double-click on the `vsim_gui.bat file`. This will spawn the ModelSim GUI. Three wave windows are given.

The stimulus data for this simulation is held in the `\HDL\CABAC\Testbench\cabac_top\stimuli` directory and consists of three input files as listed in item 2, page 17 above.

The stimulus data files provided are extracted from the reference C model (see [Ref 1]) prior to the CABAC process.

The provided default files are extracted from running 10 frames of `foreman_qcif_30` (see input sequences) through the encoder reference code.

The simulation generates an output file:

```
\Testbench\cabac_top\StiFile_bin_enc_Ecodestrm_OUT_FIFO.txt
```

The user may analyze the signals in this simulation to get a better idea of how the CABAC core should interface.

The system verification process, by which stimulus and expected results are generated from the reference code, is described in the Chapter 5. The generated files can be used in place of the default simulation files provided with the release, but must reside in the locations mentioned above during simulation.

It should be noted that the above procedure, while complex and cumbersome, is automated to become the Level 1 verification process.

XILINX®

Chapter 5

System Verification

This chapter describes the different levels of verification undergone by the CABAC encoder core. Ultimately, the system is verified by using long regression tests. The output of the hardware from these tests must exactly match the output given by the reference software which runs with the same stimulus. Two levels of testing are introduced. They vary in terms of throughput and ease of debugging.

Verification Platform Release

For running, viewing, and understanding the verification process, the release provides the following files:

1. Libraries of object (precompiled) source (same as Simulation):

   ```
   \Testbench\cabac_top\work\
   \Testbench\cabac_top\memxlib\
   \Testbench\cabac_top\mpeg4xlib\
   \Testbench\cabac_top\Sim_tools\
   ```

2. Source video sequences

   ```
   \InputSequences\
   ```

   See ""Input Sequences" in Appendix A" for full file list

3. Virtual Socket WildCard 4 platform executables and bitstreams:

   ```
   \Platform\Wildcard\test\virtual_socket\virtualsocket.exe
   \Platform\Wildcard\test\virtual_socket\XC4VSX35\FF668\VSt.x86
   ```

4. Virtual Socket top-level VHDL source code:

   ```
   \Platform\Wildcard\HW\HDL\virtual_socket\vs_fifo_move_CABAC_TOP.vhd
   ```

5. Verification scripts:

   ```
   \Verification\CABAC\CABAC_Verification.pl
   ```

6. Test bench support module:

   ```
   \TestBenchSupport\TestBenchSupport.pm
   ```

7. Software reference code executable

   ```
   \Software\ArchC_Rev1\bin\lencod.exe
   ```

Running the Verification Tests

There are 20 tests that can be run by the user. Each test has varying characteristics, including varying Video formats, interlace, parameter settings, and so on. A description of the tests is given in ""Verification Test Descriptions" in Appendix A", and is also summarized in the batch scripts listed under item item 5, page 19.

Verification at all three levels is automated down to running one of these three simple, editable batch scripts given in the directory. They all contain the command line:

```
perl -I"..\..\TestBenchSupport" CABAC_Verification.pl [TestLevel][No.
Frames][Test#]

TestLevel: 1 or 2
No. Frames: between 1 and 10 inclusive
Test#: between 0 and XXXX inclusive (0 runs all tests in order from to
XXXX)
```

The user can edit the files as desired. Double clicking on the batch file invokes the appropriate test to be executed. The two levels of verification are summarized in Table 5-1 in more detail.

Table 5-1: **Verification Level Summary**

Verification Level	Reference Executable	HW Representation	Notes
Level 1	ArchC_rev**1**\bin\lencod.exe	MTI Simulation (precompiled libraries)	• **Reference** uses structurally modified reference code to generate stimulus and expected CABAC outputs for verification at CABAC level. • **Unit under test** uses same precompiled object code libraries as simulation. Uses stimulus files generated above as simulation input stimulus. • Detailed – used for debugging. • Very slow.
Level 2	ArchC_rev**1**\bin\lencod.exe	WildCard 4	• **Reference** uses structurally modified reference code to generate stimulus and expected CABAC outputs for verification at CABAC level. • **Unit under test** uses separate executable to drive the stimulus files generated above into CABAC core on WildCard 4. • Can be used to determine frame location of errors in long regression tests. • Very fast.

Verification Process (Level 1)

Figure 5-1 is a block diagram of the Level 1 Verification flow.

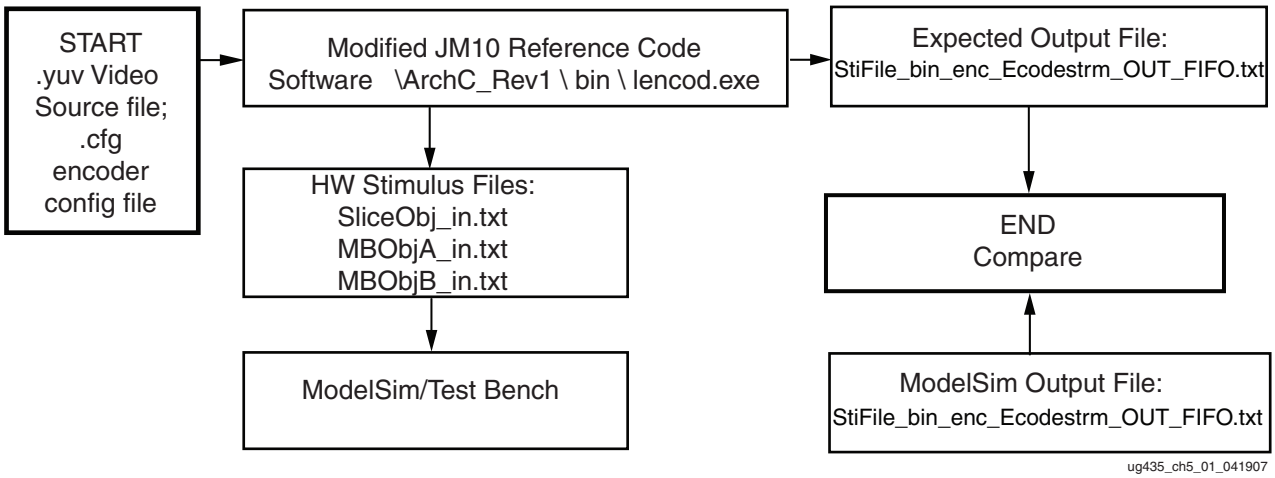

Figure 5-1: **Level 1 Verification Flow**

The command line in the batch script invokes the following processes in this order:

1. A DOS window is spawned.
2. A Perl script (`CABAC_Verification.pl`) which sets up the required test(s) in accordance with appropriate parameters is called.

   This Perl script calls functions in the general-purpose test bench support module (item 6, page 19) which runs the reference code:

   ```
   \Software\ArchC_Rev1\bin\lencod.exe
   ```

   The executable uses the video sequences in `\InputSequences` as input.
3. This code generates the stimulus files:

   ```
   \Verification\CABAC\Level1\Testxx\stimuli\SliceObj_in..txt
   \Verification\CABAC\Level1\Testxx\stimuli\MBObjA_in.txt
   \Verification\CABAC\Level1\Testxx\stimuli\MBObjB_in.txt
   ```

   …and the expected output reference file:

   ```
   \Verification\CABAC\Level1\Testxx\stimuli\XXXX.txt
   ```
4. ModelSim is called in batch mode (no GUI is invoked with the release) which generates the HW simulation output file:

   ```
   \Verification\CABAC\Level1\Testxx\XXXX.txt
   ```
5. This file is compared to the reference output file generated previously and a DOS report given. The results are also summarized in:

   ```
   \Verification\CABAC\EncoderVerification_Summary_Level1.txt
   ```

Verification Process (Level 2)

Figure 5-2 is a block diagram of the Level 2 Verification flow.

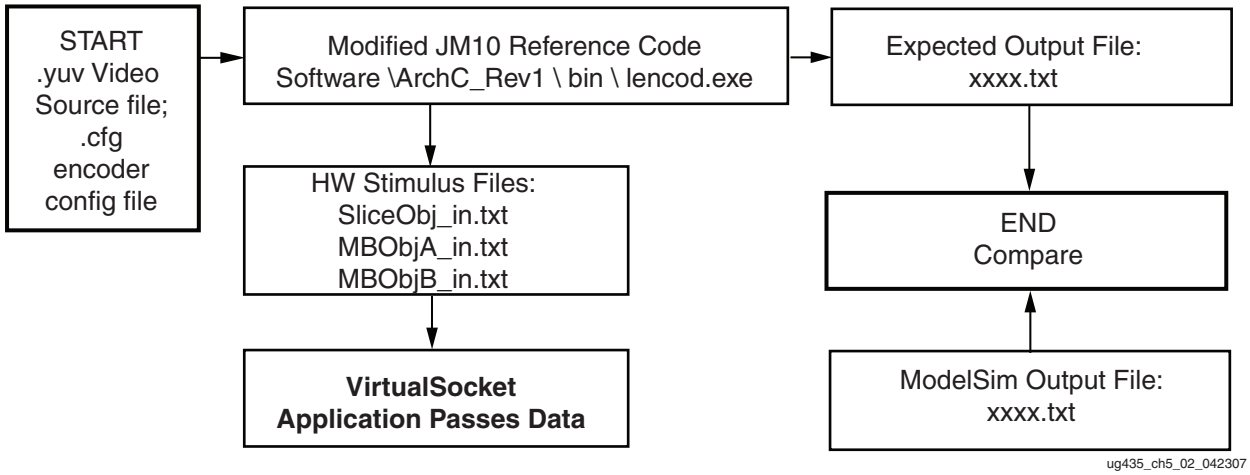

Figure 5-2: **Level 2 Verification Flow**

The command line in the batch script invokes the following processes in this order:

1. A DOS window is spawned.
2. A Perl script (`CABAC_Verification.pl`) is called, which sets up the required test(s) in accordance with appropriate parameters.
3. This Perl script calls functions in the general-purpose test bench support module (item 6, page 19) which runs the reference code:

   `\Software\ArchC_Rev1\bin\lencod.exe`
4. The executable uses the video sequences in `\InputSequences` as input.
5. This code generates the stimulus files:

   `\Verification\CABAC\Level2\Testxx\stimuli\SliceObj_in.txt`
   `\Verification\CABAC\Level2\Testxx\stimuli\MBObjA_in.txt`
   `\Verification\CABAC\Level2\Testxx\stimuli\MBObjB_in.txt`

   …and the expected output reference file:

   `\Verification\CABAC\Level2\Testxx\stimuli\XXXX.txt`
6. Another executable is called:

   `\Platform\Wildcard\Test\virtual_socket\virtualsocket.exe`
7. The WildCard 4 Hardware CABAC core generates an output file

   `\Verification\CABAC\Level2\Testxx\XXXX.txt`
8. This file is compared to the reference output file generated previously and a DOS report given. The results are also summarized in:

   `\Verification\CABAC\EncoderVerification_Summary_Level2.txt`

Verification Notes

When setting the test number to 0 in the batch script file, all tests are run in sequence from 1 to 20. When doing this, by default, all tests are run regardless of whether they have been run in the past. If some tests have already been run, but some others have not or their

results deleted, then the user may wish to run only the remaining tests. The user can do this by editing the `CABAC_Verification.pl` script, commenting out the line:

```
$ForceRegenerateAll = "1";
```

Note: Use '#' to comment.

Also, if the user wants to take this approach, but rerun one or more tests selectively, the user should delete the test directory of the test(s) to be rerun:

```
CABAC\Leveln\testxx
```

Note: *n* is the appropriate Verification Level.

and rerun the script.

XILINX®

Appendix A

Additional Information

This appendix contains the following additional information:

Input Sequences

The files provided as video input source files are:

```
\InputSequences\city_4cif_30\city_4cif_30.hdr
\InputSequences\city_4cif_30\city_4cif_30.yuv
\InputSequences\football_601i_25\football_601i_25.hdr
\InputSequences\football_601i_25\football_601i_25.yuv
\InputSequences\football_cif_30\football_cif_30.hdr
\InputSequences\football_cif_30\football_cif_30.yuv
\InputSequences\foreman_qcif_30\foreman_qcif_30.hdr
\InputSequences\foreman_qcif_30\foreman_qcif_30.yuv
\InputSequences\mobile_cif_30\mobile_cif_30.hdr
\InputSequences\mobile_cif_30\mobile_cif_30.yuv
\InputSequences\shields_720p_60\shields_720p_60.hdr
\InputSequences\shields_720p_60\shields_720p_60.yuv
\InputSequences\stockholm_1080i_30\stockholm_1080i_30.hdr
\InputSequences\stockholm_1080i_30\stockholm_1080i_30.yuv
\InputSequences\tractor_1080p_30\tractor_1080p_30.hdr
\InputSequences\tractor_1080p_30\tractor_1080p_30.yuv
```

All sequences are 10 frames long.

Directory Tree Structure

Figure A-1 shows the structure of the H.264 CABAC directories after unzipping the three ZIP files.

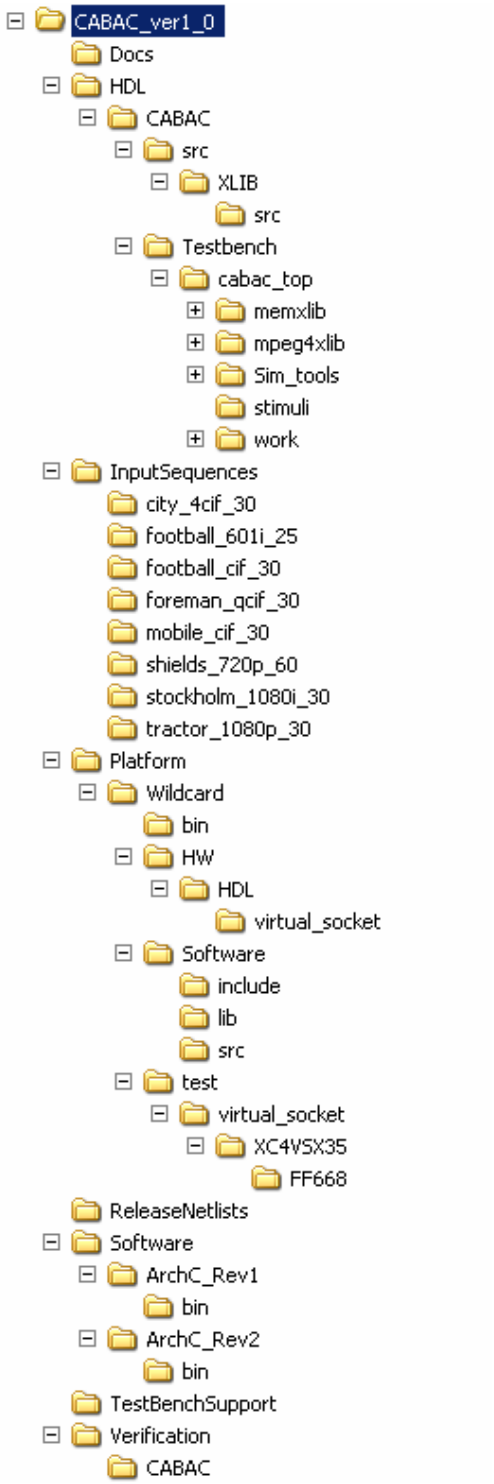

Figure A-1: **Directory Tree Structure**

Verification Test Descriptions

The default test settings are

- Rate control disabled
- Main profile
- Rate Distortion Optimization (RDO) disabled
- FixedModelNumber = 0

Table A-1 attempts to describe the regression tests used by showing their main digressions from the default settings.

Table A-1: **Regression Test Summary**

Test Number	Stream	Image Size	Bitstream	Test Specifics	Details
1	foreman_qcif_30	176 x 144	VSD_CIF	Constant QP	Rate control disabled
2	foreman_qcif_30	176 x 144	VSD_CIF	ConstantQP	High Profile, #BFrames = 2
3	foreman_qcif_30	176 x 144	VSD_CIF	Constant QP; FrameSkip	FrameSkip=2; #BFrames=2; BasicUnit=99
4	foreman_qcif_30	176 x 144	VSD_CIF	Constant QP	FrameSkip=0; #BFrames=0; BasicUnit=11
5	foreman_qcif_30	176 x 144	VSD_CIF	IntraPeriod	IntraPeriod=5; FrameSkip=0
6	foreman_qcif_30	176 x 144	VSD_CIF	Bframe	IntraPeriod=30; FrameSkip=2 #BFrames=2;
7	foreman_qcif_30	176 x 144	VSD_CIF	Init Model 1	FrameSkip=0; #BFrames=0; FixModelNumber=1
8	foreman_qcif_30	176 x 144	VSD_CIF	Init Model 2	FixedModelNumber=2
9	foreman_qcif_30	176x144	VSD_CIF	3 Slices/Frame	FrameSkip=1; #BFrames=1; SliceMode=1; SliceArgument=33
10	foreman_qcif_30	176 x 144	VSD_CIF	9 Slices/Frame	SliceMode = 1 SliceArgument = 11
11	football_601i_25	720 x 480	VSD_601	Interlace	#RefFrames=5; Picinterlace=1; SliceMode=0
12	football_601i_25	720 x 480	VSD_601	Interlace Bframes	(Default)
13	Stockholm_1080i_30	1920 x 1080	VSD_1080F	Main Profile	(Default)
14	Shields_720p_60	1280 x 720	VSD_720P	Main Profile	Main Profile
15	Tractor_1080p_30	1920 x 1080	VSD_1080	1080P	(Default)
16	city_4cif_30	704 x 576	VSD_4CIF	4CIF	(Default)
17	foreman_qcif_30	176 x 144	VSD_CIF	CIF	(Default)
18	stockholm_1080i_30	1920 x 1080	VSD_1080	1080i	Main Profile PicInterlace = 1
19	foreman_qcif_30	176 x 144	VSD-CIF	Main Profile	(Default)
20	mobile_cif_30	352x288	VSD-CIF	Main Profile	(Default)

WildCard 4 Hardware Verification Platform

The test bench was used to derive an application implemented on a WildCard 4 platform. That application centers around a module that uses the H.264 CABAC encoder core along with some additional modules to interface to the hardware located on the WildCard 4 platform. The WildCard 4 platform is a PCMCIA-based board that interfaces to a laptop. At the heart of the WildCard 4 platform is a Xilinx XC4VSX35 -10 FPGA. The encoded data is sent back to the PC via a LAD bus interface. A block diagram of that module is shown in Figure A-2. The LAD bus interface is a special interface used on the WildCard 4 platform.

Supported Hardware Platform

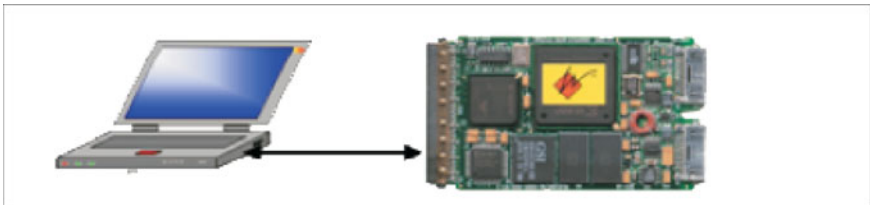

Figure A-2: **Laptop with WildCard 4 and Virtex-4 FPGA**

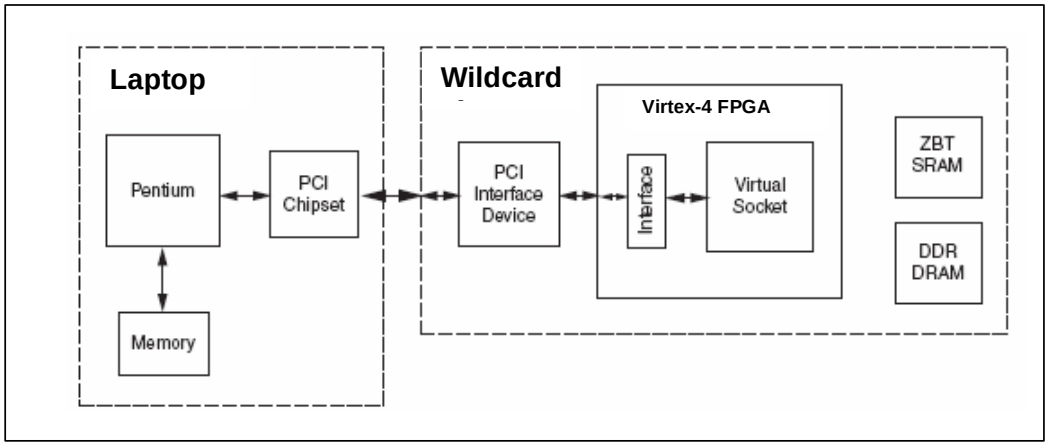

Figure A-3: **WildCard Verification System Block Diagram**

References

1. ITU-T/ISO/IEC, *Advanced Video Coding for Generic Audiovisual Services*, H.264 03/2005.