Xilinx is disclosing this Document and Intellectual Property (hereinafter “the Design”) to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED “AS IS” WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems (“High-Risk Applications”). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

© 2005-2008 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. PowerPC is a trademark of IBM, Inc. All other trademarks are the property of their respective owners.

---

**Revision History**

**MPEG-4 Simple Profile Encoder v1.2**

**UG219 (v1.2) April 14, 2008**

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/10/05</td>
<td>1.0</td>
<td>Initial Xilinx release.</td>
</tr>
<tr>
<td>12/06/05</td>
<td>1.1</td>
<td>Minor change to Table 3-3.</td>
</tr>
<tr>
<td>04/18/08</td>
<td>1.2</td>
<td>Updated for version 1.2. Added “Generating a New Bit File from a New Video Source” in Chapter 5.</td>
</tr>
</tbody>
</table>
# Contents

## Preface: About This Guide

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Resources</td>
<td>10</td>
</tr>
<tr>
<td>Acronyms Used</td>
<td>10</td>
</tr>
<tr>
<td>Conventions</td>
<td>11</td>
</tr>
<tr>
<td>Typographical</td>
<td>11</td>
</tr>
<tr>
<td>Online Document</td>
<td>12</td>
</tr>
</tbody>
</table>

## Chapter 1: Introduction

| About the Core                                | 13   |
| Features                                      | 13   |
| Applications                                  | 14   |
| Release Contents                              | 15   |
| Additional Core Resources                     | 15   |
| Technical Support                             | 15   |
| Feedback                                      | 15   |
| MPEG-4 Simple Profile Encoder Core            | 16   |
| Documentation                                 | 16   |

## Chapter 2: Getting Started

| Encoder System                                | 17   |
| Brief Package Description                     | 19   |
| Application Software                          | 19   |
| Hardware Platform                             | 20   |
| Directory Tree Structure                      | 21   |

## Chapter 3: MPEG-4 SP Encoder Core

| MPEG4_SP_Encoder Entity                       | 24   |
| Generic Definition                            | 24   |
| Port Definition                               | 26   |
| Entity Port Description                       | 27   |
| Input and Output Signals                      | 27   |
| Host                                         | 27   |
| Input                                        | 27   |
| Output                                       | 28   |
| Memory Controller                             | 29   |
| Clock and Controls                            | 29   |
| General Parameters                            | 29   |
| Motion Estimation Parameters                  | 31   |
| VLC and BP Parameters                         | 32   |
| Performance                                  | 33   |
Chapter 4: System Architecture and Interfacing

Chapter 5: MPEG-4 SP Encoder C-Model
  Running the C Model ................................................................. 37
  C Model Description .............................................................. 37
  Generating a New Bit File from a New Video Source .................. 38

Chapter 6: MPEG-4 SP Decoder C-Model
  Running the C-Model .............................................................. 41
  C Model Description .............................................................. 41

Chapter 7: Simulation Testbench
  Libraries and Testbench ....................................................... 43
    Compiled Libraries ............................................................ 43
    HDL Wrappers ................................................................. 43
    Stimuli and Expected Output Generation .............................. 45
    Simulation Run and Verification ........................................ 47

Chapter 8: WildCard-II Demonstrator
  Running the WildCard-II Demo .............................................. 49

Chapter 9: Regression Test
  Testbench Directory Creation .............................................. 51
Schedule of Figures

Chapter 1: Introduction

Chapter 2: Getting Started

Chapter 3: MPEG-4 SP Encoder Core

Chapter 4: System Architecture and Interfacing

Chapter 5: MPEG-4 SP Encoder C-Model

Chapter 6: MPEG-4 SP Decoder C-Model

Chapter 7: Simulation Testbench
Chapter 8: WildCard-II Demonstrator

Chapter 9: Regression Test

Figure 9-1: Testbench Directory Structure ................................................. 51
Figure 9-2: Directory Structure of Each Test Case. ..................................... 52

- THIS IS A DISCONTINUED IP CORE -
Schedule of Tables

Table 1-1: Acronyms Used ................................................................. 10

Chapter 1: Introduction

Chapter 2: Getting Started

Table 2-1: Software Applications ....................................................... 19

Chapter 3: MPEG-4 SP Encoder Core

Table 3-1: MPEG4_SP_Encoder Entity Generic Detailed Description .......... 25

Chapter 4: System Architecture and Interfacing

Chapter 5: MPEG-4 SP Encoder C-Model

Chapter 6: MPEG-4 SP Decoder C-Model

Chapter 7: Simulation Testbench

Chapter 8: WildCard-II Demonstrator

Chapter 9: Regression Test
Preface

About This Guide

This user guide describes the MPEG-4 Simple Profile Encoder core and a suite of applications that support it. Release 1 provided netlists at different resolutions for the user. In addition, this document supplies a number of programs that interface to that core. One program is a simple wrapper that treats the MPEG-4 encoder as a black box and adds some necessary directives for the design. This document also supplies a system-level design that shows the core in an application environment. And it provides files that can use the core inside of the testbench to help verify system operation. Finally, the document presents a demonstration of the core that uses the WildCard-II hardware platform. A regression test program is supplied to help verify proper operation of the encoder core as well as support programs like the MPEG-4 decoder C model which is useful in testing overall system operation.

Contents

This guide contains the following chapters:

- **Preface, “About This Guide”** introduces the organization and purpose of the design guide, a list of additional resources, and the conventions and abbreviations used in this document.
- **Chapter 1, “Introduction,”** describes the core and related information, including additional resources, technical support, and submitting feedback to Xilinx.
- **Chapter 2, “Getting Started,”** describes how an MPEG-4 Simple Encoder core can be included into the next hierarchy of the system architecture.
- **Chapter 3, “MPEG-4 SP Encoder Core,”** describes the overall architecture of the MPEG-4 Simple Profile Encoder core and the features shown in simulations and demonstrations.
- **Chapter 4, “System Architecture and Interfacing,”** describes the overall system architecture and how to interface to the MPEG4 Encoder core.
- **Chapter 5, “MPEG-4 SP Encoder C-Model,”** describes using the provided C-based software encoder model to help verify proper operation of the overall system design as well as functionality.
- **Chapter 6, “MPEG-4 SP Decoder C-Model,”** describes using the provided C-based software decoder model to help verify proper operation of the overall system design as well as functionality.
- **Chapter 7, “Simulation Testbench,”** describes performing VHDL simulation of the MPEG4 Encoder core.
- **Chapter 8, “WildCard-II Demonstrator,”** describes how to demonstrate a working FPGA implementation of the MPEG4 Encoder core with the WildCard-II platform.
- THIS IS A DISCONTINUED IP CORE -

Preface: About This Guide

- Chapter 9, “Regression Test,” contains information about performing regression testing to verify the operation of the MPEG-4 Simple Encoder core.

Additional Resources

For additional information, go to http://www.xilinx.com/support. The following table lists some of the resources you can access from this website or by using the provided URLs.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description/URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tutorials</td>
<td>Tutorials covering Xilinx design flows, from design entry to verification and debugging <a href="http://www.xilinx.com/support/techsup/tutorials/index.htm">http://www.xilinx.com/support/techsup/tutorials/index.htm</a></td>
</tr>
<tr>
<td>Data Sheets</td>
<td>Device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging <a href="http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp">http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp</a></td>
</tr>
<tr>
<td>Problem Solvers</td>
<td>Interactive tools that allow you to troubleshoot your design issues <a href="http://www.xilinx.com/support/troubleshoot/psolvers.htm">http://www.xilinx.com/support/troubleshoot/psolvers.htm</a></td>
</tr>
<tr>
<td>Tech Tips</td>
<td>Latest news, design tips, and patch information for the Xilinx design environment <a href="http://www.xilinx.com/xlnx/xil_tt_home.jsp">http://www.xilinx.com/xlnx/xil_tt_home.jsp</a></td>
</tr>
</tbody>
</table>

Acronyms Used

This document uses the following acronyms:

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP</td>
<td>Bitstream Packetization</td>
</tr>
<tr>
<td>CBR</td>
<td>Constant Bit Rate</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out (queue)</td>
</tr>
<tr>
<td>IC</td>
<td>Input Controller</td>
</tr>
<tr>
<td>MAD</td>
<td>Maximum Absolute Difference</td>
</tr>
<tr>
<td>MB</td>
<td>Macroblock</td>
</tr>
<tr>
<td>MC</td>
<td>Motion Compensation</td>
</tr>
<tr>
<td>ME</td>
<td>Motion Estimation</td>
</tr>
<tr>
<td>MPEG</td>
<td>Motion Picture Expert Group</td>
</tr>
<tr>
<td>MPEG-4 SP</td>
<td>MPEG-4 Simple Profile</td>
</tr>
</tbody>
</table>
Table 1-1: Acronyms Used (Continued)

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAD</td>
<td>Sum of Absolute Differences</td>
</tr>
<tr>
<td>SO</td>
<td>Software Orchestrator</td>
</tr>
<tr>
<td>TC</td>
<td>Texture Coding</td>
</tr>
<tr>
<td>TU</td>
<td>Texture Update</td>
</tr>
<tr>
<td>VLC</td>
<td>Variable Length Coding</td>
</tr>
<tr>
<td>VRB</td>
<td>Variable Bit Rate</td>
</tr>
</tbody>
</table>

Conventions

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

Typographical

The following typographical conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning or Use</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courier font</td>
<td>Messages, prompts, and program files that the system displays</td>
<td>speed grade: - 100</td>
</tr>
<tr>
<td>Courier bold</td>
<td>Literal commands you enter in a syntactical statement</td>
<td>ngdbuild design_name</td>
</tr>
<tr>
<td>Italic font</td>
<td>Variables in a syntax statement for which you must supply values</td>
<td>See the Development System Reference Guide for more information.</td>
</tr>
<tr>
<td></td>
<td>References to other manuals</td>
<td>See the User Guide for details.</td>
</tr>
<tr>
<td></td>
<td>Emphasis in text</td>
<td>If a wire is drawn so that it overlaps the pin of a symbol, the two nets are not connected.</td>
</tr>
<tr>
<td>Dark Shading</td>
<td>Items that are not supported or reserved</td>
<td>This feature is not supported</td>
</tr>
<tr>
<td>Square brackets</td>
<td>An optional entry or parameter. However, in bus specifications, such as bus[7:0], they are required.</td>
<td>ngdbuild [option_name] design_name</td>
</tr>
<tr>
<td>Braces</td>
<td>A list of items from which you must choose one or more</td>
<td>lowpwr ={on</td>
</tr>
<tr>
<td>Vertical bar</td>
<td>Separates items in a list of choices</td>
<td>lowpwr ={on</td>
</tr>
</tbody>
</table>
### Online Document

The following linking conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning or Use</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue text</td>
<td>Cross-reference link to a location in the current document</td>
<td>See the section “Additional Resources” for details. Refer to “Title Formats” in Chapter 1 for details.</td>
</tr>
<tr>
<td>Blue, underlined text</td>
<td>Hyperlink to a website (URL)</td>
<td>Go to <a href="http://www.xilinx.com">http://www.xilinx.com</a> for the latest speed files.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This document describes the MPEG-4 Simple Profile Encoder core and provides details about the core functionality and how to use it. The information encompasses modules that interface to the MPEG-4 Encoder core in a variety of ways:

- A system-level design module that interfaces the MPEG-4 Simple Profile Encoder core to a memory controller, a memory, and an input camera controller.
- A testbench module that combines that system design into an architecture that compares the results of the MPEG-4 Encoder core to a stored expected results file.
- A system application design module that uses the MPEG-4 Encoder core inside a hardware platform interfacing the decoder core to a PC.

This document defines the input and output structure of the MPEG-4 Encoder core, as well as describes and presents some VHDL testbench code, which uses the core to describe how to instantiate the core into user-specific designs. The testbench supplied shows how the core can interface to external modules. It also discusses some of the C programs and workspaces written that support designing with the MPEG-4 Encoder core.

About the Core

The aim of the MPEG-4 Encoder core is to perform the encoding of one or more input sequences into MPEG-4 simple profile bitstreams, as defined by the MPEG-4 standard ISO/IEC 14496-2, “Information Technology – Coding of Audio-Visual Objects – Part 2: Visual Amendment 1: Visual extensions,” January 2000. This core only deals with the video encoding aspects of the MPEG-4 standard, because audio is not currently included in this core.

Figure 1-1 provides a functional view of an MPEG-4 Video Encoder and the macroblock structure containing six 8x8 blocks (4 Luminance Y and 2 Chrominance U,V).

Features

The MPEG-4 Simple Profile Encoder core includes the following features:

- Half-pixel resolution adaptive directional square search with early stop criteria
- Half-pixel resolution motion compensation
- To enable constant bit rate coding, the core requires frame-based interaction with an external controller (processor, microcontroller), which calculates and updates the next frame parameters.
To enable variable bit rate coding, the controller is not required because the parameters can be set to a predetermined value. The core supports resolutions up to 720 pixels by 576 lines at 30 frames per second. Release 1 contains three different variations of the MPEG-4 decoder core based on frame resolution, QCIF, CIF, and 4CIF.

Applications

The encoder can be used in a number of potential applications that include: entertainment video; conversational H.32x service; video conferencing, streaming services; Internet and video messaging; and cellular phone networks.
Release Contents

This section provides a list of numerous items presented in this document. If the user has some or all of the necessary software support and tools, the user will have the capability to integrate the resources presented in this package into his own specific design environment to develop a video-based product centered on the simple profile of the MPEG-4 compression standard. Even without the full suite of support tools, the user will have a core that implements the MPEG-4 algorithm and a host of informative design files and programs to help guide the design development. The major sections of information about this core covered in this document include the following:

- Description of the MPEG-4 Encoder
- System architecture of an MPEG-4 Encoder
- A testbench for the system architecture that will help understand the encoder
- A description of an implementation of the encoder on a hardware platform, the WildCard-II board designed by Annapolis Micro Systems, Inc.
- A set of tools that will test the encoder with a host of compressed test images
- A description of how to use the C-Model program that implements the encoder and decoder presented with this release
- Source code information for the MPEG-4 encoder

Additional Core Resources

For detailed information and updates about the MPEG-4 Simple Profile Encoder core, see the following documents, located on the MPEG-4 Simple Profile Encoder product page at http://www.xilinx.com/ipcenter/video_codecs_index.htm.

- MPEG-4 Simple Profile Encoder Data Sheet
- MPEG-4 Simple Profile Encoder Release Notes

For updates to this document, see the MPEG-4 Simple Profile Encoder User Guide, also located on the MPEG-4 Simple Profile Encoder product page.

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the MPEG-4 Simple Profile Encoder core.

Xilinx will provide technical support for use of this product as described in this guide. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the MPEG-4 Simple Profile Encoder core and the accompanying documentation.
MPEG-4 Simple Profile Encoder Core

For comments or suggestions about the MPEG-4 Simple Profile Encoder core, please submit a webcase from www.xilinx.com/support. Be sure to include the following information:

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

Documentation

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
Chapter 2

Getting Started

Encoder System

The Encoder presented in (Figure 2-1) shows the major functions of the design. Figure 2-1 helps the user to understand and integrate the MPEG-4 encoder core into a functional system architecture. The element labeled MPEG4_encoder is the encoder core.

This MPEG-4 encoder core uses half-pixel resolution motion estimation of the incoming frame from the previous reconstructed frame, compensates it, and the residual is DCT transformed, variable length encoded and packetized. A 16x16 macroblock (MB) is the basic memory unit used in an MPEG-4 system and is defined as a block of memory that has 384 samples. For YUV 4:2:0 video data format, it contains four luma 8x8 pixel blocks, one U chroma channel 8x8 block, and one V chroma channel 8x8 block. This MPEG-4 encoder core supports 12-bit DCT coefficients (both before and after de-quantization) and 8-bit pixel inputs. It also has the capabilities to handle multiple input sequences (in Release 2), so it can be used in a number of multi-stream applications such as video surveillance/security and video conferencing.

Figure 2-1: MPEG-4 Simple Profile Encoder Functional Diagram

This MPEG-4 encoder core uses half-pixel resolution motion estimation of the incoming frame from the previous reconstructed frame, compensates it, and the residual is DCT transformed, variable length encoded and packetized. A 16x16 macroblock (MB) is the basic memory unit used in an MPEG-4 system and is defined as a block of memory that has 384 samples. For YUV 4:2:0 video data format, it contains four luma 8x8 pixel blocks, one U chroma channel 8x8 block, and one V chroma channel 8x8 block. This MPEG-4 encoder core supports 12-bit DCT coefficients (both before and after de-quantization) and 8-bit pixel inputs. It also has the capabilities to handle multiple input sequences (in Release 2), so it can be used in a number of multi-stream applications such as video surveillance/security and video conferencing.
Figure 2-1 separates the input and output signals of the MPEG-4 encoder core into functional sections. The details of the signals are presented in the MPEG-4 Simple Profile Encoder Core Specification (DS511).

![MPEG-4 Multi-Stream Encoder Functional Diagram](image-url)

Figure 2-2: MPEG-4 Multi-Stream Encoder Functional Diagram

The encoder core receives the input samples from a camera or storage interface element. The MPEG-4 Encoder accepts line-scanned macroblocks in YUV 4:2:0 format, 8 bits per pixel. A preprocessing block is required to filter and prepare data from encoding, as well convert video frames from the camera native format (e.g., RGB 4:4:4 format) to YUV 4:2:0 and make the conversion from camera scan format (e.g., line-scan) to macroblock input format required by the MPEG-4 encoder. The preprocessing block can optionally implement input image improvements that can improve the compression performance, such as de-noising, filtering, or any user-defined operation that removes unwanted data.

The memory interface section of the MPEG-4 encoder core attaches to an external memory controller via multiple ports on the codec in either the read or the write mode. The memory controller module is the hardware that allocates which port has access to the memory at a given instance in time. The interface to/from the memory controller occurs in a burst mode to simplify interface considerations and speed up memory access times. This control information overhead is transmitted to the controller at power up and is a system-defined parameter.

The input frames are encoded and packetized by the MPEG-4 encoder and each packet is delivered to the transmitter in bytes. The encoder also outputs as parameters the number of bits produced for the current packet. The parameter interface is used to signal that the encoder has processed the current frame, and to communicate to the bit-rate controller the size of the packet used to calculate the parameters for the next frame.

The encoder can work in stand-alone mode for Variable Bit Rate (VRB). However, for Constant Bit Rate (CBR), it requires a controller that implements a frame-based bit rate...
control with one frame delay, since the number of bits is not known until after the frame is encoded. For VBR (stand-alone mode), the encoder parameters are set up front and not changed during processing. In the CBR mode, the encoder parameters are potentially changed every frame to keep the encoding bit rate constant. The bit rate control algorithm is implemented on a general-purpose controller (e.g., MicroBlaze™, PowerPC™, host processor) and is delayed by one frame.

The control section of the system architecture, not included in Figure 2-1, is the clock generation circuitry as well as global reset.

**Brief Package Description**

This package contains application software, a hardware verification and demonstration platform, and details of the directory structure.

**Application Software**

The modules supplied with this platform release use a variety of application programs. Your interest dictates what applications are necessary to carry out user tasks. The complete sets of applications used with the modules supplied by this release are listed below.

**Table 2-1: Software Applications**

<table>
<thead>
<tr>
<th>Program/Utility</th>
<th>Version</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelSim</td>
<td>5.8c SE, 5.7f PE</td>
<td>Simulation Environment</td>
</tr>
<tr>
<td>Synplicity Synplify Pro</td>
<td>7.5.1, 7.7.1</td>
<td>VHDL Synthesizer</td>
</tr>
<tr>
<td>Xilinx ISE™</td>
<td>6.2.03i</td>
<td>Place and Route Tools</td>
</tr>
<tr>
<td>Microsoft Visual Studio C++</td>
<td>6.0</td>
<td>Software Development Platform</td>
</tr>
<tr>
<td>ActivePerl</td>
<td>5.8.3</td>
<td>Interpreter used for regression test</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(<a href="http://www.activestate.com/Products/ActivePerl/">http://www.activestate.com/Products/ActivePerl/</a>)</td>
</tr>
<tr>
<td>Annapolis Micro</td>
<td>API: 2.6, Driver: 3.3, Firmware: 1.6</td>
<td>Hardware Verification Platform</td>
</tr>
<tr>
<td>Systems WildCard-II Platform</td>
<td></td>
<td>(<a href="http://www.annapmicro.com/">http://www.annapmicro.com/</a>)</td>
</tr>
</tbody>
</table>
Hardware Platform

A useful extension to this core release is the WildCard-II board from Annapolis Microsystems, Inc. Annapolis Microsystems has boards that can be inserted into a laptop computer and demonstrate the MPEG-4 encoder core in near real-time. This allows for demonstration and verification of the actual hardware running on the FPGA. The current PC board contained within the WildCard-II system configuration contains a Xilinx XC2V3000 –4 FPGA, ZBT memory, and an interface to the PCMCI bus.

Figure 2-3: WildCard-II Hardware Verification Platform Block Diagram

To demonstrate the MPEG-4 encoder, a system was built that interfaces to this hardware platform under the control of the microprocessor on the laptop (see Figure 2-1). There is a precompiled executable that controls a variety of test and display features to verify the encoder hardware operating on the WildCard-II. One function of the program is the loading of the design programming bit file that was created for the XC2V3000 device located on the hardware platform. That bit file houses a number of modules along with the MPEG-4 encoder that performs video compression from the encompassed YUV file presented to it. The compressed bitstream is then passed back to the PC’s processor to be stored and verified.
Directory Tree Structure

Figure 2-4 gives the user the exact structure of the release directory. Important files that are discussed in this manual reside in the structure presented in these figures. Figure 2-4 shows the entire tree structure of the MPEG-4 SP Encoder core release starting at the top level and showing many of the subdirectories.

The EDIF netlist for the encoder is contained in the ReleaseNetlists directory. The MPEG-4 Simple Profile encoder is available in two main flavors: a core without external memory controller and a core including the external memory controller for 2 Mbyte ZBT SRAM. The content of the ReleaseNetlists is shown in the Figure 2-5. The naming convention represents the most important parameters of the generated EDIF netlist: MPEG4_SP_ENCODER_<Technology>_MAXIMUM_RESOLUTION_<Memory controller included>.edf. Each netlist file is provided with two associated files that represents the typical HDL instantiation template (<netlist>.vho) and a text file containing a detailed list of generic parameters of the core and their value used to generate the respective core. The ZBT SRAM memory controller files are located in the HDL library directory.
Figure 2-5: Release Netlists Directory Structure
Chapter 3

MPEG-4 SP Encoder Core

The MPEG-4 SP encoder takes its input as YUV macroblocks through encoder input interface, represented in Figure 3-1 as ENC_IN*. The encoder produces the encoded bitstream through the output interface, represented in Figure 3-1 as ENC_OUT*. The encoder also produces a parameter, which represents the number of bits produced for each frame encoded. The parameter is produced after the last byte of the current encoded frame (each frame generates a parameter). It can be used by the user as system synchronizations because it comes at well-known moments in time and is required by the rate control algorithm (its main purpose). This parameter needs to be read when the encoder is used in streaming mode and no rate control enabled by this parameter can be ignored. The parameter interface is represented in Figure 3-1 as ENC_PARAMS*. The encoder requires two ports to a memory controller, one read port and one write port, represented in Figure 3-1 as MEM_WQ*. The encoder also has a host interface for setting the core in the desired configuration and read processing statistics for rate control algorithm. The host interface is represented in Figure 3-1 as HOST_*.

The MPEG-4 encoder core has a number of generic parameters that are selected to create the netlists to control maximum horizontal and vertical resolution, input buffer size and bit width, and output buffer size and bit width. The generics section of the entity wrapper also selects the type of memory to be used in the local buffer on the FPGA.
MPEG4_SP_Encoder Entity

The entity block showing the generic section of the wrapper entity, MPEG4_SP_Encoder, is presented for 4CIF resolution:

**Generic Definition**

```vhdl
generic (
  MAX_WIDTH : integer := 720;
  MAX_HEIGHT : integer := 540;
  MAX_PACKET_SIZE : integer := 2048;
  SMALL_MEM_TYPE : string := "select_ram";
  INPUT_BUFFER_SIZE : integer := 4096;
  OUTPUT_BITWIDTH : integer := 32 -- only 32 or 8
);
```

**Figure 3-1: MPEG-4 SP Encoder Schematic Symbol**

---

**- THIS IS A DISCONTINUED IP CORE -**
Table 3-1 shows the MPEG4_SP_Encoder entity generic detailed description.

**Table 3-1: MPEG4_SP_Encoder Entity Generic Detailed Description**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_WIDTH</td>
<td>Integer</td>
<td>Maximum width of frame</td>
</tr>
<tr>
<td>MAX_HEIGHT</td>
<td>Integer</td>
<td>Maximum height of frame</td>
</tr>
<tr>
<td>MAX_PACKET_SIZE</td>
<td>Integer</td>
<td>Maximum video packet size</td>
</tr>
<tr>
<td>SMALL_MEM_TYPE</td>
<td>String</td>
<td>Type of the small memory: “select_ram”- distributed ram “no_rw_check”- block RAM (speed) “block_ram, area” – block RAM (area)</td>
</tr>
<tr>
<td>INPUT_BUFFER_SIZE</td>
<td>Integer</td>
<td>Input Buffer Size</td>
</tr>
<tr>
<td>INPUT_BITWIDTH</td>
<td>Integer</td>
<td>Input Bit Width 32 or 8</td>
</tr>
<tr>
<td>OUTPUT_BUFFER_SIZE</td>
<td>Integer</td>
<td>Output Buffer Size</td>
</tr>
<tr>
<td>OUTPUT_BITWIDTH</td>
<td>Integer</td>
<td>Output Bit Width 32 or 8</td>
</tr>
</tbody>
</table>

The user can instantiate a core with different input and output types. The input interface is defined by two generics: (a) INPUT_BUFFER_SIZE, which defines the input buffer size and INPUT_BITWIDTH, which defines the input bit width. Only two values are allowed for the input bit widths, 8 and 32. The input interfaces and additional logic instantiate are represented in Figure 3-2. In this release, only 32-bit width is supported. Depending on what generic the user uses, it will make a structure such as the one described in Figure 3-2 and Figure 3-3.

**Figure 3-2: MPEG-4 SP Encoder Input Interface Types: (a) 8 Bit (b) 32 Bit**

The output interface is defined by two generics (a) OUTPUT_BUFFER_SIZE, which defines the output buffer size and OUTPUT_BITWIDTH, which defines the output bit width. Only two values are allowed for the output bit width, 8 and 32. The output interfaces and additional logic instantiate are represented in Figure 3-3. The flush signal is used to make sure that all data that has been produced by the encoder is put into the output buffer. The signal is used only in the 32-bit instantiation of the output interface. In this release, only 32-bit width is supported.
Port Definition

The port definition section of the MPEG-4 Encoder, which also helps to define the signals that are related to the MPEG-4 encoder core, follows:

```vhdl
port(
    reset              : in std_logic;
    clk                : in std_logic;

    -- External Interface
    host_ce                 : in std_logic; -- chip enable
    host_we                 : in std_logic; -- write
    host_re                 : in std_logic; -- read
    host_re_ack             : out std_logic; -- read acknowledge
    host_address            : in std_logic_vector(4 downto 0);
    host_din                : in std_logic_vector(31 downto 0);
    host_dout               : out std_logic_vector(31 downto 0);

    -- Interface MB_FIFO
    enc_in_empty      : out std_logic;
    enc_in_full       : out std_logic;
    enc_in_we         : in std_logic;
    enc_in_d          : in std_logic_vector(INPUT_BITWIDTH-1 downto 0);

    -- Interface outputBuffer
    enc_out_full : out std_logic;
    enc_out_empty : out std_logic;
    enc_out_re    : in std_logic;
    enc_out_q     : out std_logic_vector(OUTPUT_BITWIDTH-1 downto 0);

    -- Interface parametersOut
    enc_params_full : out std_logic;
    enc_params_empty : out std_logic;
    enc_params_re : in std_logic;
    enc_params_q : out std_logic_vector(29 downto 0);

    -- Interface recFrameRead
    MEM_RQ_FLUSH           : out std_logic;
    MEM_RQ_JOB_PUSH        : out std_logic;
    MEM_RQ_JOB_FULL        : in  std_logic;
    MEM_RQ_JOB_ERR         : in  std_logic;
    MEM_RQ_JOB_AD          : out std_logic_vector(31 downto 0);
    MEM_RQ_POP             : out std_logic;
    MEM_RQ_EMPTY           : in  std_logic;
    MEM_RQ_AE              : in  std_logic;
    MEM_RQ_RCNT            : in  std_logic_vector(6 downto 0);
    MEM_RQ_RERR            : in  std_logic;
    MEM_RQ_RD              : in  std_logic_vector(31 downto 0);

    -- Interface recFrameWrite
    MEM_WQ_FLUSH           : out std_logic;
    MEM_WQ_PUSH            : out std_logic;
    MEM_WQ_FULL            : in  std_logic;
)
```

Figure 3-3: MPEG-4 SP Encoder Output Interfaces Types: (a) 8 Bit (b) 32 Bit

(a) (b) ug219_06_102605

- THIS IS A DISCONTINUED IP CORE -
MEM_WQ_AFULL : in std_logic;
MEM_WQ_WCNT : in std_logic_vector(6 downto 0);
MEM_WQ_WERR : in std_logic;
MEM_WQ_WAD : out std_logic_vector(31 downto 0);
MEM_WQ_WBX : out std_logic_vector(4 downto 0);

-- Added for flushing 32 bit word at end of sequence
word_flush : in std_logic
);
end MPEG4_SP_Encoder;

Entity Port Description

The MPEG-4 Simple Profile Encoder data sheet (DS511) presents the signal descriptions of the MPEG4_SP_Encoder entity ports.

Input and Output Signals

The MPEG-4 encoder module has input and output signals that can be broken into subsections and interface to a specific module. The major sections of the encoder are:

- Host
- Input
- Output
- Memory Controller
- Clock and Control

*Note:* All MPEG-4 encoder core signals operate on the positive edge of the clock and are Active High unless explicitly stated.

Host

The host interface provides the user or encoder's driving (embedded) system the opportunity to change the parameters for processing the next frame and to receive the statistics used by the rate control algorithm for calculating the new processing parameters. The encoder uses the parameters at the beginning of the frame and cannot be changed during processing of the current frame. The user is responsible to provide in time the updated parameter based on the synchronization possibilities offered by the output interface.

The MPEG4 Simple Profile Encoder data sheet (DS511) presents the writing and reading timing diagram of the MPEG-4 Simple Profile encoder host interface. The host_ce (chip enable) is used to enable the operations on the interface. If the signal is not active all consequent operations are ignored. The writing operation requires that host_we (write enable) is active and the data from host_din (data input) is written into the register file of the encoder at the address present on the host_address (address) port. The reading operation requires that host_re (read enable) is active and the date from the address present on host_address (address) is valid next rising edge of the clock on host_dout (data out). The address mapping and the content significance are presented in Table 17 of the MPEG-4 Simple Profile Encoder data sheet (DS511).

Input

The input interface requires the captured macroblocks to be presented in a strict sequence. Any inconsistency between the number of pixels input, the order, and the parameter can
desynchronize the encoding process. The input macroblocks have to be line scanned one-
by-one left to right. Each macroblock (Y, U, & V) is presented at the input as planar
scanned. YUV on macroblock basis (16×16 Y, 8×8 U, 8×8 V). The input pixels are ordered
according to the numbering presented in Figure 3-4.

The macroblock’s input interface is defined as a typical FIFO interface as described in
Figure 3-4. The control signal enc_in_full indicates that the input FIFO is full and
cannot accept more data, and enc_in_empty indicates that there is no more data in the
queue. The control signal enc_in_we (write enable) indicates that the data present on port
enc_in_d (data in) will be pushed into the queue. The size of the input buffer can be set
as a generic on the component instantiation and its default value is 511. The empty signal
of the FIFO is provided to the user to enable fast and efficient burst write operations up to
the size of the input buffer.

Output
The output interface is composed out of two FIFO interfaces, one buffering the bitstream
and the other buffering the parameter output of the MPEG-4 SP encoder. Both are defined
as typical FIFO interfaces. The control signal enc_out_full indicates that the input
FIFO is full and cannot accept more data, and enc_out_empty indicates that there is no
more data in the queue. The control signal enc_out_re (read enable) indicates that data
will be popped from the queue and the next clock cycle will be valid on port enc_in_d
(data out). The size of the output buffer can be set as a generic on the component
instantiation and its default value is 511 for the netlists provided in Release 1. The full
signal of the FIFO is provided to the user to enable fast and efficient burst read operations
up to the size of the output buffer. The size of the parameter buffer is fixed to 15. The
parameter output buffer contains an element for every fully processed frame, and it
contains the number of information bits produced for the current frame. This can be used
as an interrupt for the higher level system to trigger the bit rate control algorithm to
calculate the new encoding parameter for the next frame, efficiently transfer the data from
the output buffer (data size is known), and load the next input frame.

Memory Controller

The memory controller interface stores and retrieves portions of a frame’s worth of video
information for the encoder’s motion compensation and motion estimation component.
Previously processed macroblocks are an important component in the motion
compensated MPEG-4 encoder. The encoder has the capability to store a little more than a
line’s worth of macroblocks on the chip to speed up the encoding process. The memory
interface allows data to go to and from an external memory via a memory controller
external to the MPEG-4 encoder core. A ZBT memory controller is contained with one
of the supplied system configuration designs that is in a testbench test application and the
WildCard-II application. That memory controller handles up to four bidirectional data
paths. There is a write queue and a read queue in the memory controller and WQ* and RQ*
interface signals to those respective FIFOs.

In the write queue, there is a 32-bit data word interface to the memory controller and
ultimately to the memory as identified by WQ0_WAD. There is an associated 5-bit control
bus with those data words. Writing to the queue is performed by the WQ0_PUSH signal in
a burst mode that is defined for that data transfer. Error and status information signals are
provided as needed to help direct the flow of information. The data sheet (DS511) shows
interaction between the write port of the memory controller and the MPEG-4 encoder core.

The read queue looks very similar to the write queue where the direction of the flow of
information is into the encoder. The data sheet shows the interactions of the read portion of
the memory controller. Note that the address is presented on RQ0_JOB_AD when the
RQ0_JOB_FULL line is not active and is identified with the RQ0_JOB_PUSH line active.
The signal RQ0_AE triggers the reading process causing the RQ0_POP signal to go active
for a burst size and the data presented on the RQ0_RD bus.

Clock and Controls

The clock and controls section of the encoder incorporates two input signals: CLK, SCLR.
A CLK signal has to be supplied to the encoder for operation. The signal should be a fifty-
percent duty cycle, and all operations inside the encoder are processed on the rising edge
of the clock.

General Parameters

The vopRoundingType defines the type of the currently encoded frame: Intra coded
frame or inter frame predicted.

The frameMemoryOffset is the parameter that enables multiple streams encoder. The
encoder processes one frame at a time (slices for the encoder are fixed to frames), and it
uses a reference frame in the processing. The number of streams that the encoder can
process is defined by the memory size that stores the reference frames as illustrated in Figure 3-5.

![Figure 3-5: External Memory Organization for Multiple Streams: Two Streams Example (one QCIF and one CIF)](image)

The QP parameter defines the quantization parameter used by the texture encoding chain. It can have values from 1 to 32.

The frameMBHeight and frameMBWidth parameters define the height and the width in macroblocks. One macroblock has the dimension of 16×16.
Motion Estimation Parameters

The motion estimation algorithm implemented in the MPEG-4 SP core is a squared directional search with early stop criteria. The algorithm has the following stages:

1. Motion Vector Prediction – The start point of the motion estimation is predicted based on the motion vectors from the surrounding macroblocks for which motion has been previously estimated.

2. Predicted Position – the sum of absolute difference at the predicted position.

3. First Square Search – the sum of absolute difference (SAD) is calculated for surrounding positions of the predicted motion vector. In this step, eight locations are evaluated.

4. Directional Search Step – the search for maximum absolute difference is continued in the direction of the SAD improvement. In this step, three locations are evaluated. This step is repeated until the early stop criteria is met or the maximum spiral number is exceeded.

5. Half PEL Motion Estimation – for the best location found in the previous steps, the half PEL motion estimation is performed by interpolating the reference frame.

At any of steps 2 to 4, the search can be stopped by the early stop criteria:

- if the search yields not improvement in term of SAD
- if the current SAD are under a predefined reference SAD, which signifies that the current match is sufficiently good for current search

At any of steps 2 to 4, a decision can be taken for the current macroblock to be coded as an intra-macroblock if a good match is not found. See Figure 3-6 for the motion estimation algorithm.

![Motion Estimation Algorithm: Squared Directional Search](image_url)
The algorithm is directional because the search for improvements is made in the direction of the best motion vector found. It is adaptive because the number of searches is limited by the early stop criteria, which stops the algorithm as soon as a good match is found.

The motion estimation also generates a set of statistics which characterizes the movement and the information that is to be encoded. This information is used by the Rate Control algorithm to calculate the parameter for the next frame.

The `maxSpiralNumber` defines the maximum number of spirals the search can extend. The typical numbers of spirals is related to the resolution: 2 for resolutions below QCIF and 3 for higher resolutions.

The `vopRoundingType` defines the type of rounding used in the interpolation filter for half PEL search phase. If the user activates rounding control, the rounding type is defined as:

- for I frame: `vopRoundingType = initialValue`
- else: `vopRoundingType = vopRoundingType XOR 1`

The `SADref` is the SAD that defines the threshold which triggers the early stop criteria. This means that the match is found and the SAD of the block is smaller than the `SADref`, then it is a high probability that the difference information will be trimmed by the quantization process. There is direct connection between the `QP` parameter and `SADref`:

\[
SADref = 256 + 70 \cdot QP
\]

The `MAD` is a statistics gathered by the ME block and it represents the Maximum Absolute Difference of the current frame.

The `SAD_NoMotion` is a statistic gathered by the ME block. It represents the Sum of Absolute Difference of the all blocks encoded as intra blocks (with no motion vectors) in the current frame.

The `SAD_Motion` is a statistic gathered by the ME block. It represents the Sum of Absolute Difference of the all blocks encoded as inter blocks (with motion vectors) in the current frame.

**VLC and BP Parameters**

The `dataPartitioning` parameter indicates whether or not the data partitioning is enabled. This version of the encoder does not support data partitioning. The flag should always be disabled.

The `errorResilienceDisable` parameter indicates whether or not the error resilience is enabled. This version of the encoder does not support data partitioning. The flag should always be disabled.

The `targetPacketSize` parameter defines the targeted size of a packet. In the data partitioning configuration, a packet is related to a slice, and the VLC automatically ends a slice when the target packet size is exceeded. Because the data partitioning is not supported in this version, the role of the target packet size represents the buffering between VLC and BP.

The `DCvlcThres` parameter is a user specified code that allows a mechanism to switch between two VLCs for coding of intra DC coefficients.

The `moduloTimeIndex` is a 1-bit flag that indicates when more than one second absolute time has elapsed since the previous frame.
The `timeIncrementResolution` parameter indicates the number of evenly spaced subintervals, called ticks, within one modulo time. One modulo time represents the fixed interval of one second. It is defined as the smallest 16-bit unsigned integer $x$ meeting:

$$\left\lfloor \frac{x}{f_e} + 10^{-4} \right\rfloor < 10^{-4}$$

with $f_e$ the encoding frame rate.

The `timeIncrementWidth` specifies the bit width of `timeIncrementResolution`.

The `timeIncrement` parameter represents the absolute time increment from the synchronization point marked by the `modulo_time_base` measured in the number of clock ticks.

The `quantPrecision` parameter represents the number of bits required to represent the quantization parameter $Q_P$.

The `videoObjectID` parameter represents the Video object ID of the currently encoded video sequence (used when processing multiple streams).

The `generateVolHeader` parameter indicates to the BP to generate a new VOL header. This flag can be used to force the insertion of the VOL header, which automatically ends the previous stream. This can be used as means of reformatting on the fly (frame aligned) the basic properties of the stream. (e.g., end QCIF sequence and start 4CIF sequence). The VOL header must always be inserted in the conjunction with an Intra coded frame.

The `MBsInVOPbitWidth` parameter specifies the bit width of $frameMBwidth \times frameMBheight$.

**Performance**

The current revision of the MPEG-4 encoder operates at 70 MHz on a Virtex-II XC2V3000-6 part using Synplicity 7.5.1 and ISE 6.3.2. At this clock rate, the throughput of the current revision of the decoder is approximately 45 K MB/s (worst case). Future revisions of the encoder have improved clock frequency and throughput. A particular system configuration can achieve clock rates close to 100 MHz on Virtex-4 -12 integrated circuits thereby achieving approximately 55 K MB/s (worst case) performance. Obviously, the system configuration, input data rate, and input bitstreams will dictate actual performance.
Chapter 3: MPEG-4 SP Encoder Core

- THIS IS A DISCONTINUED IP CORE -
System Architecture and Interfacing

In this release, the designer has the MPEG-4 encoder core available in nine different EDIF files as described in the data sheet.

This section describes how the MPEG-4 encoder core can be included into the next level of the system architecture hierarchy.

This encoder core module is used to create a bigger system configuration by adding a memory controller module, memory, and preprocessing block. A diagram of the additional functional block is presented in Figure 4-1.

![Figure 4-1: MPEG-4 Encoder Core Module Block Diagram Showing the Memory Controller](https://www.xilinx.com/ug219/10_110705)
Based on these two variants of the top-level encoder in the HDL\Libraries\src\MPEG4XLIB directory, there are two VHDL wrapper files:

- MPEG-4 SP encoder without memory controller in MPEG4_SP_Encoder.vhd
- MPEG-4 SP encoder including memory controller in MPEG4_SP_Encoder_MemCtrl.vhd

The MPEG-4 encoder core uses one read channel and one write channel of the memory controller. After the macroblock has been reconstructed in the Texture_Update module, the information is sent to a local storage area to be used in motion compensation operations for future macroblocks. The information is packed into a 32-bit word format and sent to the controller in bursts along with a starting address for the burst of data. Data is retrieved from memory in a similar manner when requested by the CopyController module, which requests a macroblock of data when signaled by the Parser_VLD module. It then proceeds to store that macroblock in the local on-chip memory that has a capacity of a little more than a macroblock worth of data. The word length and burst mode of operation is similar to the write operation with the exception that data is being read from the external ZBT SRAM memory.
MPEG-4 SP Encoder C-Model

Included in the Utilities ZIP file is a software model of the MPEG4 encoder. This software allows for verification of the hardware simulation and emulation. The user can run the software in standalone mode with input YUV files to get an early evaluation of the bit rate for their application. Steps to running the software model follow.

Running the C Model

To run the MPEG-4 SP Encoder SOFTWARE demo:

1. Make sure that the `forman_qcif_30.yuv` file is in the `InputSequences\foreman_qcif_30` directory.
3. This batch file reads the `.yuv` file and produces a `.bit` file based on the control (`.ctl`) and configuration (`.cfg`) files in the `Exe` directory. The batch file runs encoder and decoder software for the stream selected. Additionally the resulting `.bit` file and reconstructed `.yuv` files are located in the `Exe` directory for review.

C Model Description

The C model program objective is to create a compressed bit file of a specified image sequence. The sequence YUV file is stored as indicated above for the QCIF foreman sequence. A user generated control file, as defined above, is also necessary to steer the C program. The C model takes the source file and control parameters as input parameters for the software that generates the needed compressed bit file. This file can then be used by the matching decoder C model to recreate the original image sequence.

The C model program for the MPEG-4 encoder uses the following directories:

- Software/MPEG4Ver2_encoder
- Software/MPEG4Ver2_encoder/encoder
- Software/MPEG4_decoder/demo
- Software/MPEG4Ver2_decoder/direct_draw_viewer
- Software/MPEG4Ver2_encoder/StimuliGeneration
- Software/MPEG4Ver2_encoder/testbench
- Software/MPEG4Ver2_encoder/exe
The workspace file is located in the first directory, the MPEG4Ver2_encoder directory (for short) and is called MPEG4Ver2_encoder.dsw. That directory also contains the AllInOne.h file that has been used by other C models and will be used again in the C model for the MPEG-4 encoder core. Observe that this file has important defines that dictate the operation of the C code and are shown below. These defines are located in the AllInOne.h file and steer the results of the C program.

... 
#define USE_DISPLAY // show reconstructed frame 
...

#define simpleME // restrict ME to 3 neighbors and no global search 
#define likeHW_ME // avoid to soon stops in comparison with HW 
#define likeHW_TC // perform (I)DCT like HW version 
#define likeHW_BP //count all bits like HW version 

The code has been designed to allow a variety of operations to perform. Therefore, there is executable code already predefined for operations needed for this release, such as the executable that creates the image sequence test stream for the regression test. These executables are located in the Release directory defined above. The decoder directory contains source code for the C model while the other directories contain additional support files.

Note: All necessary executables are provided in the Exe directory and the user can use the software at his/her own risk. Any changes to the C code require the Microsoft Visual C™ software and license. The project file (.dsw) loads all of the necessary files for compilation of the code.

Generating a New Bit File from a New Video Source

This section describes step-by-step how to generate a new MPEG-4 bitstream from a video source.

1. Find a video source, such as a web cam or stream off the Internet.
2. Create a video file that is of a CIF resolution (352x288) or known resolution which is an integer number of macroblocks.
3. Find a program that converts the given file into planer YUV format, sometimes referred to as raw I420 format.
4. Create a header (.hdr) file for your video, using the input sequences for the MPEG-4 encoder as an example. You will need the following information:
   a. Frame rate 
   b. Number of frames 
   c. Resolution 

The header file examples can be found in:
$MPEG_ROOT/InputSequences/your_sequence/sequence_name.hdr

This file is not required, but is useful because the information that you put into the file will be used when you construct the control and configuration files following. It also serves as a reference for later, since the raw files do not contain any information on the video stream.
5. Create control and configuration files, using the mobile_cif_30 configuration and control files as a starting point. You will need to modify the directories and file names to match the location of your files.

   The main parameter to change is the {VOL end frame}. Make sure that this matches the number of frames in your input video file. You might also want to reduce the {Target bit-rate for VOL} to between 800k and 400k to reduce the overall output bit file size.

6. Create a .bat file using the format in the file MPEG-4_Codec.mobile_cif.bat in the following directory as an example (MPEG-4 Encoder release utilities ZIP file).

   $MPEG_ROOT/Software/Exe

7. Run the batch file to generate the bitstream with the MPEG-4 encoder software model. Decode with the MPEG-4 decoder software model.
Chapter 6

MPEG-4 SP Decoder C-Model

Running the C-Model

To run the MPEG-4 SP Decoder SOFTWARE demo:
1. The decoder software can be run by executing with a control file:
   \Software\Exe\MPEG4Ver2_decoder_ddraw.exe <ctl_file>
The control file must point to the bit file to be decoded.
2. There are .bat files located in the Software\Exe directory which illustrate encoding and decoding with the software models.

C Model Description

The software accepts compressed image bitstreams selected by the Microsoft C Studio environment. A number of image sequences have been stored to be viewed by a viewing application, as well as the compressed bitstream of a variety of images at different set parameters.

The C model program for the MPEG-4 decoder uses the following directories:

MPEG4Ver2_decoder/MPEG4_decoder
MPEG4Ver2_decoder/MPEG4_decoder/decoder
MPEG4Ver2_decoder/MPEG4_decoder/debug
MPEG4Ver2_decoder/MPEG4_decoder/demo
MPEG4Ver2_decoder/MPEG4_decoder/direct_draw_viewer
MPEG4Ver2_decoder/MPEG4_decoder/exe

The workspace file is located in the first directory, the MPEG4_decoder directory (for short) and is called MPEG4Ver2_decoder.dsw. That directory also contains the AllInOne.h file that has been used by other C models and will be used again in the C model for the MPEG-4 decoder core. Observe that this file has important defines that dictate the operation of the C code and are shown below. These defines are located in the AllInOne.h file and steer the results of the C program.

//#define VERBOSE // enables more detailed information during run time
#define STOP_ON_ERROR
//#define USE_DISPLAY_GDI // uses the gdi display mode
#define USE_DISPLAY_DIRECTX // uses directx
#define WRITE_FILE // used for the regression test
#define VIEW_STATISTICS  // important statistical information

// Only one type of viewer allowed
#ifdef USE_DISPLAY_DIRECTX
#undef USE_DISPLAY_GDI
#endif

The code has been designed to allow a variety of operations for the user to perform. Therefore, there is executable code already predefined for operations needed for this release, such as the executable that creates the image sequence test stream for the regression test. These executables are located in the Exe directory defined above. The decoder directory contains source code for the C model while the other directories contain additional support files.

**Note:** All necessary executables are provided in the exe directory and the user can use the software at his/her own risk. Any changes to the C code require the Visual C software and license.
Chapter 7

Simulation Testbench

A simulation model is provided for ModelSim simulation. The compiled libraries are provided together with a testbench and software model to generate the stimuli and the expected output for the testbench. The simulation libraries are located in the MPEG-4 SP Encoder package at the HDL location.

Libraries and Testbench

Compiled Libraries

The user is provided with the ModelSim compiled libraries of the encoder and all supporting libraries. The compiled libraries are located at HDL\Libraries\compiled. The content of the compiled libraries directory is presented in Figure 7-1.

HDL Wrappers

The user is provided with two HDL wrapper files that allows the simulation of the two main versions the MPEG-4 Encoder: with the memory controller included in the core or without the memory controller included in the core. The HDL wrappers are located at: HDL\Libraries\src\MPEG4XLib and are named MPEG4_SP_Encoder.vhd for the version without the memory controller and MPEG4_SP_Encoder_MemCtrl.vhd for the version including the memory controller.

Each encoder version is instantiated in a testbench and linked with the files that contain the input stimuli. The general testbench architecture is represented in Figure 7-2 for the MPEG-4 encoder without memory controller and in Figure 7-3 for the MPEG-4 Encoder with memory controller included in the testbench.
The testbench directory is located at `HDL\Simulation` and it contains a directory for each core version: `MPEG4_SP_Encoder` for the core without memory controller and `MPEG4_SP_Encoder_MemCtrl` for the one with memory controller. Both testbenches contain similar structure represented in Figure 7-4.
The directory contains:

- Analyze.bat – analyze the testbench HDL files.
- AnalyzeFiles.tcl – TCL script for analyzing the HDL files, associated with Analyze.bat
- Modelsim.ini – INI file for ModelSim, where all used library are linked to the current simulation.
- encoder_top_TB.vhd – MPEG-4 SP Encoder HDL testbench.
- vsim_all.bat – simulation executed in batch mode (no waveforms).
- vsim_gui.bat – simulation executed in GUI mode, where the following .do files are loaded to view waveforms: encoder_top_core.do for top-level signals and encoder_top_user.do for detailed internal signals.
- Stimuli – directory that contains the stimuli for current simulation.

**Note:** The user needs to make sure that the following library statements are added in the generic modelsim.ini or manually add them in the current Modelsim.ini.

**Example:**

In C:\Modeltech_6.1a\modelsim.ini add:

- simprim = C:\Xilinx\vhdl\mti_se\simprim
- unisim = C:\Xilinx\vhdl\mti_se\unisim
- XilinxCoreLib=C:\Xilinx\vhdl\mti_se\XilinxCoreLib

**Stimuli and Expected Output Generation**

The stimuli and expected output are generated based on the precompiled executable located in Software\Exe:

- MPEG4Ver2_encoder_StimuliNoMemCtrl.exe – for the MPEG-4 Encoder without memory controller.
- MPEG4Ver2_encoder_StimuliWithMemCtrl.exe – for the MPEG-4 Encoder with memory controller.

However, the user can rebuild the executable by recompiling the software model and enabling the required define in the software model,

Software\MPEG4Ver2_encoder\stimuliGeneration\ stimuliGeneration.h

```c
#define StimuliGenerationCompleteEncoder //Encoder without MemCtrl
#define StimuliGenerationCompleteEncoder_top //Encoder with MemCtrl
```
The user needs to recompile and copy the executable to the Exe directory and rename it appropriately.

In the stimuli directory of each testbench, there is one batch file for each of the three sequences in the InputSequences directory, together with the associated *.cfg (configuration file) and *_enc.ctl (encoder control files).

The number of frames generated by the model, together with other encoder parameters can be changed in the *.cfg file.

An example configuration file is given below. (For more information on configuration files, refer to ISO/IEC 14496-2, “Information Technology - Coding of Audio-Visual Objects - Part 2: Visual, Amendment 1: Visual extensions,” January 2000.):

```
0 {VO Id}
1 {No. of layers: 1-single layer coding, >1-scalable coding}
0 {VOL Id}
30 {VOL frame rate}
1 {Value of M (M-1-no. of B-VOPs between two consecutive P-VOPs)}
176 {Hor. size of Y image on disk}
144 {Hor. size of Y image on disk}
30 {Frame rate of source sequences on disk}
0 {Alpha channel: 0-rect., 1-binary, 2-grey level, 3-bin. shape cod. only}
1 {OBMC disable: 0-enable, 1-disable}
5 {quantizer precision}
8 {bits per pixel}
.
.
foreman_qcif_30.bit {Name of bitstream file for this VOL}
0 {VOL start frame}
9 {VOL end frame}
0 {Time (ms) at which VOL first appears}
20000 {Target bit-rate for VOL}
```

In this example, only the first 10 frames are written into stimuli files. In the stimuli directory by running the *.bat file, the software model generates following files:

- For the core without memory controller:
  - IC_InputSequence_tb_ref.in.txt - input sequence file
  - ConfigRAM_tb_ref.in.txt - input host configuration file
  - BPoutput_tb_ref.out.txt - expected output bitstream file
  - BPparametersOut_tb_ref.out.txt - expected output bitstream file
  - CCrecFrame_tb_ref.in.txt - Memory controller operation for read port file
  - TUrecFrame_tb_ref.out.txt - Memory controller operation for write port file

- For the core with memory controller:
  - IC_InputSequence_tb_ref.in.txt - input sequence file
  - ConfigRAM_tb_re.in.txt - input host configuration file
  - BPoutput_tb_ref.out.txt - expected output bitstream file
  - BPparametersOut_tb_ref.out.txt - expected output bitstream file
Simulation Run and Verification

To run the simulation:

1. Generate the stimuli by running the appropriate batch file on the
   HDL/Simulation/MPEG4_SP_ENCODER/stimuli location.

2. Links to the external libraries (UNISIM, SIMPRIM and XilinxCoreLib) should be
   inserted into the modelsim.ini file on your system.

3. Run Analyze.bat to analyze the testbench.

4. Choose the simulation type by running:
   a. Vsim_all.bat – batch mode
   b. Vsim_gui.bat – for GUI mode and execute run –all

5. Verify the results by running the comp.bat.

6. Review the results on the log.txt or by reviewing the timing diagram generated.

For debug purposes, or if there are differences, the user can compare the simulation output
with the expected output from the Stimuli directory. The simulation generates the
following files in the simulation directory:

- BPoutput_tb_dat.out.txt – expected output bitstream file
- BPparametersOut_tb_dat.out.txt – this is the output generated by the
  simulation.

**Note:** For a high number of frames, the simulation time can be very long. Recommended number of
frames for reasonable simulation time is 10 QCIF frames or equivalent.

The verification comp.bat generates a log.txt file in which the simulation output is
compared with the expected output.
- THIS IS A DISCONTINUED IP CORE -

Chapter 7: Simulation Testbench

- THIS IS A DISCONTINUED IP CORE -
Running the WildCard-II Demo

To run the Encoder hardware demo provided for the WildCard-II platform, perform the following steps:

1. Install the WildCard-II software.
2. Insert a WildCard-II into a PCMCIA slot.
3. From a Windows Explorer window, double-click on: MPEG-4_SP_Demo.bat
4. A transcript window appears on the desktop with a top-level debug menu. In this window, choose your stream by typing type 'number' (1, 4, 7, 20) and <Enter>
5. Watch the video window that pops up and make note at end of run for performance characteristics listed in transcript window. You may have to move your command window to see the video sequence.
6. Repeat steps 3 to 5 as desired to run different sequences.

The display feature of the WildCard-II demonstration displays the incoming .yuv sequence. To review the compressed output, the user is directed to run the decoder software to view the display and generate the compressed output .yuv file.
Chapter 8: WildCard-II Demonstrator

- THIS IS A DISCONTINUED IP CORE -
Regression Test

To ensure that the MPEG-4 encoder module is performing properly, the current release has included a C program workspace as well as an executable file that verifies the operation of the MPEG-4 encoder core. The software references a predefined number (3) of stored image sequences, ranging from QCIF to 4CIF resolutions, and compares those files to ones produced by the MPEG-4 core implemented on the WildCard-II hardware platform. The regression test determines whether or not the expected results match the output of the core and identifies any potential errors for each image sequence.

To perform the regression test, the set of images used to compare to the one generated by the core have to be created and stored in the appropriate directory. There is a batch file named `run_enc_v1.bat` that calls a Perl script file that creates the three sequences in the appropriate directory. The batch file is located in the `Testbench` directory.

Testbench Directory Creation

The purpose of the testbench is to generate the reference encoded sequences for the hardware encoder. The testbench has been defined based on software profiling and covers all the functionality of the encoder. The testbench files are also used to generate hardware-encoded bitstreams and compare this result with the reference bitstreams to check their correctness. The same testbench is used as regression test by running and comparing all defined sequences.

The testbench contains three encoded sequences that are defined in the Perl script `run_enc_v1.pl`. Based on the test list and three additional Perl scripts `default_cfg.pl`, `default_enc_ctl.pl`, `default_dec_ctl.pl`, all directories and all required files are created. Figure 9-1 shows the list of all files contained in the testbench directory after the Perl script has been run by the wrapping batch file `run_enc_v1.bat`.

![Figure 9-1: Testbench Directory Structure](image)

- THIS IS A DISCONTINUED IP CORE -
Running the `run_enc_v1.bat` creates for each sequence a directory structure similar to the one in Figure 9-2, where the sequence configuration file is generated (`<sequence name>.cfg`) together with the associated encoding and decoding control files (`<sequence name>_enc.ctl` and `<sequence name>_dec.ctl`). These files are used by the software model to generate the reference streams in `data_enc_refer_hw` and by the actual hardware implementation of the MPEG-4 Simple Profile Encoder in `data_enc_test_hw`.

![Directory Structure of Each Test Case](image)

**Figure 9-2:** Directory Structure of Each Test Case