MPEG-4 Simple Profile Decoder v1.3

User Guide

UG211 (v1.3) April 14, 2008

XILINX®

Revision History

MPEG-4 Simple Profile Decoder v1.3
UG211 (v1.3) April 14, 2008

The following table shows the revision history for this document.

Date	Version	Revision
10/24/05	1.1	Initial Xilinx release.
12/05/07	1.2	Added “Generating a New Bit File from a New Video Source”section to Appendix D.
04/14/08	1.3	Updated core version to 1.3. Minor edits.

Contents

Schedule of Figures

Schedule of Tables

About This Guide

The MPEG-4 Simple Profile Decoder User Guide describes the basic function and operation of the Xilinx LogiCORE™ MPEG-4 Simple Profile Decoder core. It contains information about designing, customizing, and implementing the core. It also describes some of the very useful tools that have been supplied with the release that were used on an earlier version of the decoder core. This means that some of the functions presented in this document may not be supported in the current version of the decoder core, but will be supported in future versions. In any event, they are useful tools that will assist you in the overall operation of the MPEG-4-based decoder.

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 used in this document.
- Chapter 1, "Introduction" describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Installing the Core" provides information about installing and licensing options the core.
- Chapter 3, "Basic MPEG-4 Simple Decoder"describes the overall architecture of the MPEG-4 Simple Profile Decoder core, and the features shown in simulations and demonstrations that are not supported in the current MPEG-4 Simple Profile release and its corresponding data sheet.
- Chapter 4, "Designing with the MPEG-4 Simple Profile Decoder Core" describes how an MPEG-4 Simple Decoder core can be included into the next hierarchy of the system architecture.
- Chapter 5, "Implementing Your Design" provides information for using the provided test bench to generate, compare, and implement the results of your design.
- Appendix A, "Related Information"This appendix references related useful information and material not otherwise included with MPEG-4 Simple Profile Decoder core documentation. This includes links to web pages for trade groups, specifications, articles, etc.
- Appendix B, "Directory Tree Structure" provides information about the structure of the MPEG-4 Simple Profile Decoder directory and the location of files.
- Appendix C, "Verification and Interoperability" contains information about performing regression testing to verify the operation of the MPEG-4 Simple Decoder core codec.

- Appendix D, "Using the MPEG-4 SP Decoder and Encoder C-Models" describes using the provided C-based software decoder and encoder models to help verify proper operation of the overall system design as well as specific sub-section functionality.

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.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://www.xilinx.com/support/techsup/tutorials/index.htm
Answer Browser	Database of Xilinx solution records http://www.xilinx.com/xlnx/xil_ans_browser.jsp
Application Notes	Descriptions of device-specific design techniques and approaches http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Application+Notes
Data Sheets	Device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues http://www.xilinx.com/support/troubleshoot/psolvers.htm
Tech Tips	Latest news, design tips, and patch information for the Xilinx design environment http://www.xilinx.com/xlnx/xil_tt_home.jsp

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 you enter in a syntactical statement	**`ngdbuild`** `design_name`

Convention	Meaning or Use	Example
Italic font	Variables in a syntax statement for which you must supply values	See the *Development System Reference Guide* for more information.
	References to other manuals	See the *User Guide* for details.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **bus[7:0]**, they are required.	**ngdbuild** [option_name] design_name
Braces { }	A list of items from which you must choose one or more	**lowpwr ={on|off}**
Vertical bar |	Separates items in a list of choices	**lowpwr ={on|off}**
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Omitted repetitive material	**allow block** block_name loc1 loc2 ... locn;
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	usr_teof_n is active low.

Online Document

The following linking conventions are used in this document:

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

XILINX®

Chapter 1

Introduction

This chapter introduces the MPEG-4 Simple Profile Decoder core and provides related information, including recommended design experience, additional resources, technical support, and ways to submit feedback to Xilinx. This guide is intended to give the user a working "sandbox" to understand and implement video codec sub-systems.

About the Core

The MPEG-4 Simple Decoder core can decode an MPEG-4 simple profile bitstream as defined by the MPEG-4 standard (see reference 1). This core deals with only the video decoding aspects of the MPEG-4 standard. Audio is not currently supported in this core.

It is important to understand that the programs referred to in this document, and used to create the demonstrations on the Wildcard II platform use the same basic functions presented in the MPEG-4 Simple Profile v1.1 Decoder Core Product Specification, but have slightly different interfaces. System configurations were designed using that building block to demonstrate simulation capabilities and real-time operation. The MPEG-4 Simple Decoder core v1.1 does not include all the functionality presented in this user guide. The interfaces of the MPEG-4 Simple Decoder core were modified to make it more user-friendly.

Some features, such as multi-stream processing have been presented for the interest of the user, but are not supported in this release of the MPEG-4 Simple Decoder core. In addition, some of the items referred to in this document refer to source code that is not part of the current release. The descriptions are provided to help you understand the features presented in this guide.

The Decoder core can potentially be used in a number of applications that include entertainment video, conversational H.32x service, video conferencing, streaming services, Internet and video messaging, and cellular phone networking.

The MPEG-4 Simple Profile Decoder core is a Xilinx CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see http://www.xilinx.com/ipcenter/video_codecs_index.htm. For information about system requirements, installation, and licensing options, see Chapter 2, "Installing the Core."

The major sections of information about this core that are covered in this user guide include the following:

- Description of the MPEG-4 Simple Decoder codec.
- Exemplary System Architecture that uses a MPEG-4 Simple Profile Decoder.
- A test bench for the system architecture to help understand the codec.

- A description of an implementation of the codec on a hardware platform, the WildCard-II board designed by Annapolis Micro Systems Inc.
- A set of tools to test the codec with a host of compressed test images.
- A description of how to use the C-Model program that implements the decoder and the encoder presented with this release.

Recommended Design Experience

Although the MPEG-4 Simple Profile Decoder 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 user constraints files (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 MPEG-4 Simple Profile Decoder core, see the following documents, located on the MPEG-4 Simple Profile Decoder product page at http://www.xilinx.com/ipcenter/video_codecs_index.htm.

- *MPEG-4 Simple Profile Decoder Data Sheet*
- *MPEG-4 Simple Profile Decoder Release Notes*

For updates to this document, see the *MPEG-4 Simple Profile Decoder User Guide*, also located on the MPEG-4 Simple Profile Decoder 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 Decoder 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 Decoder core and the accompanying documentation.

MPEG-4 Simple Profile Decoder Core

For comments or suggestions about the MPEG-4 Simple Profile Decoder 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

Document

For comments or suggestions about this document, please 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

Installing the Core

This chapter provides instructions for installing the MPEG-4 Simple Profile Decoder core. The MPEG-4 Decoder core 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.

System Requirements

Windows

- Windows® 2000 Professional with Service Pack 2 or greater
- Windows XP Professional Service Pack 1 or greater

Software

- Xilinx ISE 6.3i with Service Pack 3

  If necessary, ISE 6.3i Service Packs can be downloaded from http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp?software=x.xx.
- ModelSim 5.8c SE
- MicroSoft Visual Studio C++ 6.0
- ActivePerl 5.8.3

Hardware

- Anapolis Micro Systems Wildcard II Platform
  - API 2.6
  - Driver 3.3
  - Firmware 1.6

Installing the Core

Install the core by performing a manual installation after downloading the core from the web.

Manual Installation

1. Download the IP Update ZIP file from the following location and save it to a temporary directory:

   http://www.xilinx.com/ipcenter/video_codecs/mpeg4_registration.htm

  - If prompted to enter a login name and password, enter your Xilinx login name and password.
  - If you are new to Xilinx, click Create an Account and follow the instructions.
2. Unpack the ZIP file using WinZip 7.0 SR-1 or later.
3. Extract the ZIP file (MPEG4_SP_Decoder_ver1_0.zip) archive to the root directory of your Xilinx software installation. Allow the extractor utility you use to overwrite all existing files and maintain the directory structure defined in the archive.

Chapter 3

Basic MPEG-4 Simple Decoder

This chapter describes the overall architecture of the MPEG-4 Simple Profile Decoder core.

The basic building blocks are used in many places within this document and in the supported netlist files. They have been used to create a simulation environment to help you understand the MPEG-4 Decoder core, and to assist you in developing a demonstration of core functionality. For example, a feature not currently supported but presented in this document is the multistream decoding environment.

Detailed information about the supported features of the MPEG-4 Decoder core are presented in the MPEG-4 Simple Profile Decoder Data Sheet, which can be found in the `MPEG_SP_decoder/doc/hdl_specs` directory of this release.

MPEG-4 Simple Profile Multi-stream Decoder

A block diagram of the MPEG-4 Simple Decoder core is shown in Figure 3-1. A number of input FIFOs are generated by user directives inside the core. The FIFOs are there to support multistream processing, are not included in this release of the MPEG-4 Decoder core. The number of FIFOs generated depends on the number of streams within the design, as directed by a generic parameter. Each macroblock is processed one block at a time within the decoder hardware block. A complete video frame is processed before the decoder goes to another stream for processing.

The parser/VLD synchronizes to the data and extracts motion vector values, and AC and DC coefficient values. It then sends the information to the proper module. It also sends an address to a memory read block that performs a macroblock look-ahead function.

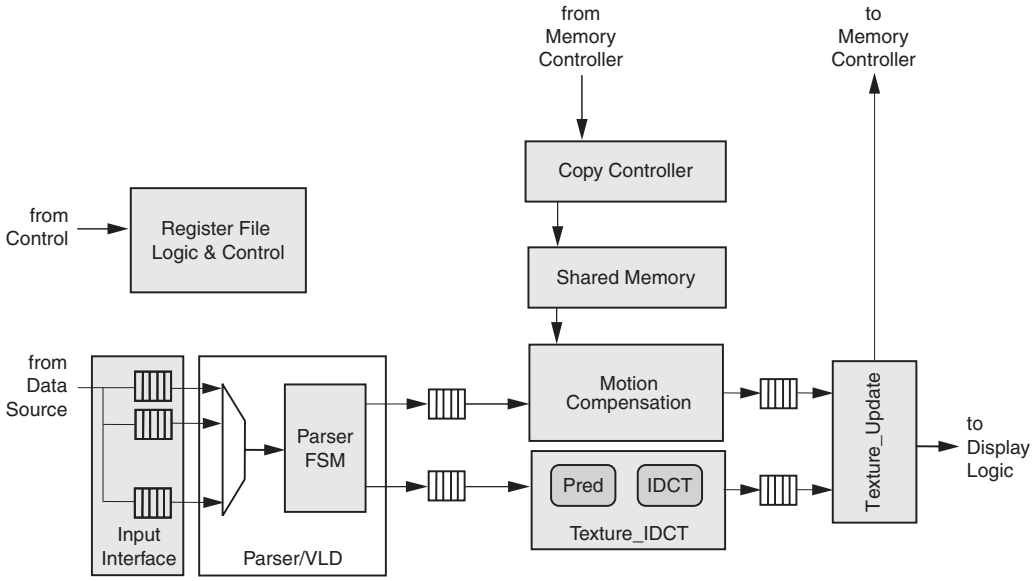

Figure 3-1: **MPEG-4 Simple Decoder Block Diagram**

The shared memory holds slightly more than a single line of macroblocks and is used to implement the motion compensation algorithm without waiting for long memory accessing times.

A schematic symbol of the MPEG-4 Simple Profile Decoder core is presented in Figure 3-2 with labels for all of the inputs and outputs of the MPEG-4 Simple Decoder core codec used in the Widlcard-II platform.

Note that this symbol is not the same symbol used in the MPEG-4 Simple Profile Decoder v1.3 Product Specification. This system does not include a higher-level wrapper that adjusts some signal polarities as well as signal names.

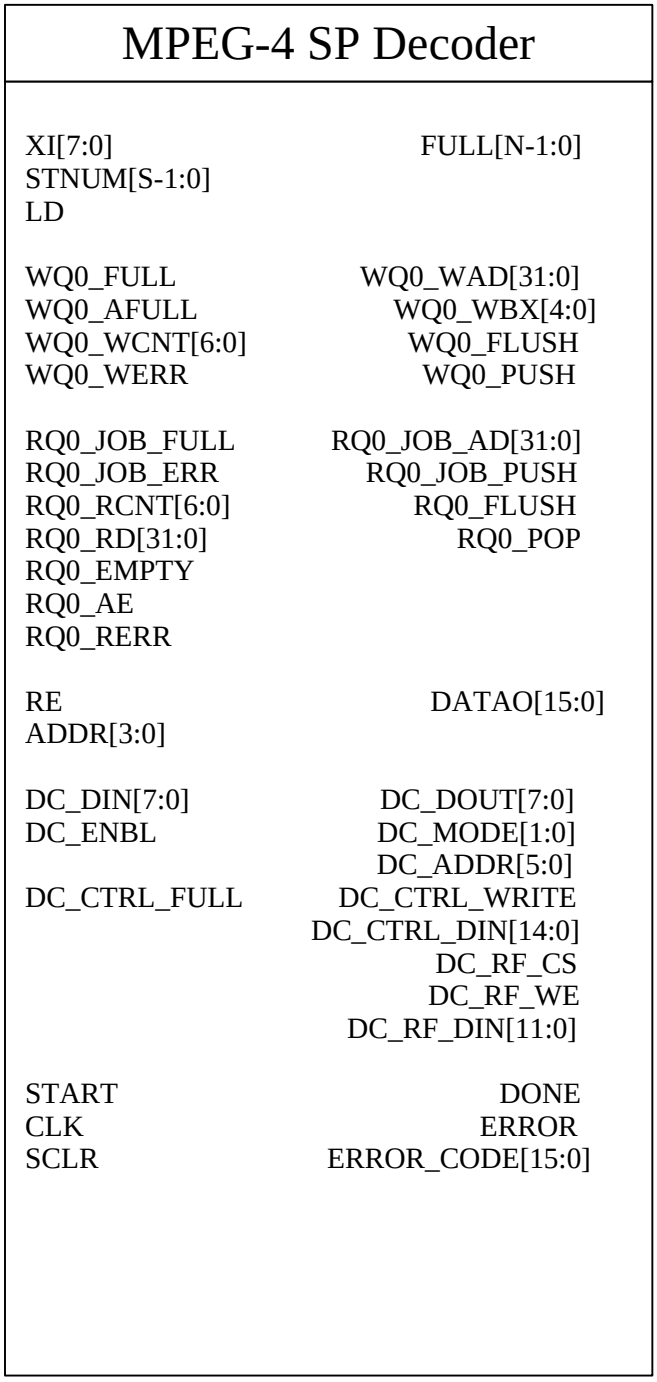

Figure 3-2: **MPEG-4 Wildcard-II Platform Symbol**

A special version of the MPEG-4 Decoder core has been provided to test a specific core implementation using the Wildcard-II environment. The EDIF file for this core is bundled with the top level EDIF file and assembled using the black box synthesizer directives. The executable batch file is located in the Wildcar-II directory that performs this function.

MPEG-4 Simple Decoder Subsections

The MPEG-4 Simple Decoder module has a number of input and output signals that can be broken into subsections and then interfaced to a specific module.

The major sections of the Decoder core are:

- Input interface
- Memory Interface
- Display Controller
- Control

Additional outputs have been made available for monitoring and test operations, based on user preferences and options desired for individual designs. All signals of the MPEG-4 Decoder core operate on the positive edge of the clock, and are active high unless otherwise stated.

Input Interface

The input section of the Decoder core connects to the entity that carries the compressed bitstream to the decoder. A tuner derives the compressed bitstream and presents that 8-bit data word to the Decoder core on the XI input bus and the stream number it is associated with on the STNUM bus. The FULL signal signifies that the FIFO for the identified stream is either ready, or not able to accept more information. The LD signal from the external tuner device will transfer the information into the decoder on the next clock edge whenever active.

The interface to the display controller is the vehicle to present the uncompressed video sequence to the display medium in YUV 4:2:0 format. These signals start with the prefix "DC_". DC_ENBL tells the decoder that the display controller is ready for data to be placed on the DC_DOUT data bus. The 2-bit DC_MODE signal and the DC_ADDR data bus are signals to the display controller input FIFO sections. Recall that the transfer occurs in bursts due to the nature of macroblock processing. DC_DIN [7:0] is not used. In addition to the data, control signals are passed to the display controller using DC_CTRL_FULL, DC_CTRL_WRITE and DC_CTRL_DIN[14:0] to define the current block number and X, Y position.

DC_RF_CS, DC_RF_WE, and DC_CTRLDIN [11:0] convey image size and memory address information.

Control

The clock and controls section of the decoder incorporates three input signals: CLK, SCLR, and START. A CLK signal has to be supplied to the MPEG-4 Decoder core, which then uses it to operate. The signal should have a fifty-percent duty cycle, and all operations inside the Decoder core are processed on the rising edge of the clock. The reset signal to the Decoder core is the SCLR (synchronous clear) signal. The START signal is important because it initiates the decoding operation and must be active to maintain the decoding operation. The output signals are ERROR, DONE and ERROR_CODE. Whenever an error situation occurs, it is identified on this signal. The additional 16-bit error code bus supplies more details about the error condition. Table 3-1 provides a list of error codes and their respective descriptions.

Table 3-1: **Error Code Detection Mapping**

ERROR_CODE[7:0]	Description
0x00	No error found
0x01	VO/VOL start codes not found
0x02	VOP start code not found (resynchronization issued)
0x03	Illegal or unsupported parameter in VOL header
0x04	Error coding MCBPC in MB header
0x05	Inter MB found in I-VOP
0x06	Error reading CBPY code
0x07	Error position BB-1 in DCT variable length decoding
0x08 – 0xFF	Reserved for future use

Memory Interface

The memory controller interface stores and retrieves portions of a single frame of video information for the motion compensation component of the MPEG-4 Decoder core. Previously processed macroblocks are an important component in a motion compensated decoder. The MPEG-4 Decoder core can store a little more than a line's worth of macroblocks on the chip to speed up the decoding process. The memory interface allows data to go to and from an external memory using a memory controller external to the core. A ZBT memory controller is contained with one of the supplied system configuration designs in a test bench test application, and with the Wildcard-II application. That memory controller handles up to four bi-directional data paths. A write queue and a read queue are contained within the memory controller and WQ* and RQ* interface signals to those respective FIFOs.

Write Queue

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 `WQ0_PUSH` in a burst mode defined for that data transfer. Error and status information signals are provided as needed to help direct the flow of information.

Read Queue

The read queue looks very similar to the write queue where the direction of the flow of information is into the MPEG-4 Decoder core. The interactions of the read portion of the memory controller is presented below. 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 `RQ0_AE` signal triggers the reading process causing `RQ0_POP` to go active for a burst size and the data presented on the `RQ0_RD` bus.

The port definition section of the MPEG-4 Simple Decoder core is presented below. It also helps to define the signals that are related to the MPEG-4 Simple Decoder core. These are the lower-level port definitions for the released MPEG-4 Simple Decoder core that have been surrounded with a higher-level wrapper to simplify their use.

```
port (
--input section
xi                  :in std_logic_vector(BITS_STREAM-1 downto 0);
stnum               : in std_logic_vector(BITS_STNUM-1 downto 0);
ld                  : in std_logic;
full                : out std_logic_vector(STREAMS-1 downto 0);
 -- Interface to display controller
dc_enbl             : in std_logic;
dc_mode             : out std_logic_vector(1 downto 0);
dc_addr             : out std_logic_vector(5 downto 0);
dc_dout             : out std_logic_vector(BITS_PIXEL-1 downto 0);
dc_din              : in std_logic_vector(BITS_PIXEL-1 downto 0);
dc_ctrl_full        : in std_logic;
dc_ctrl_write       : out std_logic;
dc_ctrl_din         : out std_logic_vector(2*BITS_DIM+2 downto 0);
-- Register file interface (currently not used)
re                  : in std_logic;
addr                :in std_logic_vector(3 downto 0);
datao               :out std_logic_vector(15 downto 0);
 -- Clock and controls
start               :in std_logic;
clk                 :in std_logic;
sclr                :in std_logic;
error_code          :out std_logic_vector(15 downto 0);
error               :out std_logic;
done                :out std_logic;
Memory Controller
WQ0_FLUSH           :out std_logic;
WQ0_PUSH            :out std_logic;
WQ0_FULL            :in std_logic;
WQ0_AFULL           :in std_logic;
WQ0_WCNT            :in std_logic_vector(6 downto 0);
WQ0_WERR            :in std_logic;
```

```
port (
WQ0_WAD         :out std_logic_vector(31 downto 0);
WQ0_WBX         :out std_logic_vector(4 downto 0);
RQ0_FLUSH       :out std_logic;
RQ0_JOB_PUSH    :out std_logic;
RQ0_JOB_FULL    :in std_logic;
RQ0_JOB_ERR     :in std_logic;
RQ0_JOB_AD      :out std_logic_vector(31 downto 0);
RQ0_POP         :out std_logic;
RQ0_EMPTY       :in std_logic;
RQ0_AE          :in std_logic;
RQ0_RCNT        :in std_logic_vector(6 downto 0);
RQ0_RERR        :in std_logic;
RQ0_RD          :in std_logic_vector(31 downto 0);
```

Pre-Compiled EDIFs of MPEG-4 Simple Decoder

The MPEG-4 Simple Decoder core is presented as an EDIF file. There are 16 forms of the MPEG-4 Simple Decoder core pre-synthesized. The generic parameters have been selected and are frozen within the EDIF file. The pre-compiled file names are listed in

Table 3-2: **Pre-compiled EDIF Filenames**

File Name	Embedded Memory Controller	Resolution
MPEG4_SP_Decoder_4CIF_V4.edf	no	4CIF
MPEG4_SP_Decoder_4CIF_V4_wMEM.edf	yes	4CIF
MPEG4_SP_Decoder_CIF_V4.edf	no	CIF
MPEG4_SP_Decoder_CIF_V4_wMEM.edf	yes	CIF
MPEG4_SP_Decoder_4CIF_V2P.edf	no	4CIF
MPEG4_SP_Decoder_4CIF_V2P_wMEM.edf	yes	4CIF
MPEG4_SP_Decoder_CIF_V2P.edf	no	CIF
MPEG4_SP_Decoder_CIF_V2P_wMEM.edf	yes	CIF
MPEG4_SP_Decoder_4CIF_S3.edf	no	4CIF
MPEG4_SP_Decoder_4CIF_S3_wMEM.edf	yes	4CIF
MPEG4_SP_Decoder_CIF_S3.edf	no	CIF
MPEG4_SP_Decoder_CIF_S3_wMEM.edf	yes	CIF

Table 3-2: **Pre-compiled EDIF Filenames *(Continued)***

MPEG4_SP_Decoder_4CIF_V2.edf	no	4CIF
MPEG4_SP_Decoder_4CIF_V2_wMEM.edf	yes	4CIF
MPEG4_SP_Decoder_CIF_V2.edf	no	CIF
MPEG4_SP_Decoder_CIF_V2_wMEM.edf	yes	CIF

Latency

The current Wildcard-II revision of the MPEG-4 Simple Decoder operates at 70 MHz on a Virtex-II XC2V3000 -6 and at 84 MHz on a Virtex-4 XC4VSX25 -12 part using Synplicity 7.5.1 and ISE 6.3.2. Table 3-3 provides the approximate speed of the MPEG-4 Decoder core in terms of macroblocks per second.

Table 3-3: **MPEG-4 Simple Decoder Clock Speeds**

Clock Frequency	Macroblocks/Second (approximately)
50M	90K
60M	108K
70M	126K
80M	144K
90M	162K
100M	180K

Future revisions of the MPEG-4 Decoder core will have improved clock frequency and throughput. System configuration and input bitstreams will dictate actual performance.

Design Resources

Table 3-4 shows the number of slices, block RAMs, and multipliers allocated for the MPEG-4 Decoder core for the eight cases. This information can help you to plan for the resources you need for typical decoder configurations. Note that the resources are for the MPEG-4 Decoder core module only and do not include the memory controller or the display controller. The requested clock rate during place and route was 70 MHz.

Table 3-4: **MPEG-4 Resource Allocation**

Frame Size	Streams	BRAMS	SLICES	MULTs
352 x 288	1	16	4558	32
	2	17	4767	32
	4	19	5037	32
	8	23	5305	32
720 x 576	1	26	5004	30

Table 3-4: **MPEG-4 Resource Allocation**

	2	27	5184	31
	4	29	5473	31
	8	33	5764	32

XILINX®

Chapter 4

Designing with the MPEG-4 Simple Profile Decoder Core

This chapter describes how to include an MPEG-4 Decoder core into the next hierarchy of system architecture.

Overview

The MPEG-4 Decoder core-based architecture is presented in Figure 4-1 as a simple block diagram to help you understand the major functions of the design and to integrate the MPEG-4 Simple Decoder core into a functional system architecture.

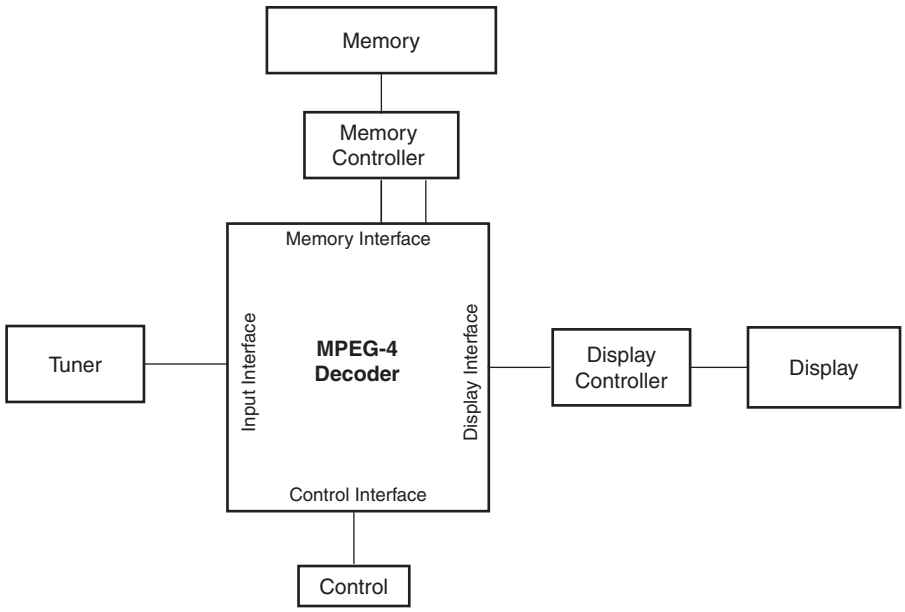

Figure 4-1: **MPEG-4 Simple Decoder Functional Diagram**

This MPEG-4 Simple Decoder core uses half-pel resolution motion compensation from the previous reconstructed frame and IDCT based residual processing. 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 Simple Decoder core supports 12-bit DCT coefficients (both before and after de-quantization) and 8-bit pixel outputs. Because it can handle multiple compressed input bitstreams, it can be used in a number of multi-stream applications such as video

surveillance, security, and video conferencing. This specific configuration of the MPEG-4 Decoder core can only support up to eight streams. Multi-streams can be implemented on the core, but have not been included in this release.

Figure 4-1 separates the input signals and output signals of the MPEG-4 Decoder core into functional sections. The signal names and description of the functions of the released MPEG-4 Decoder core are contained in the *MPEG-4 Simple Profile Decoder Core Data Sheet* and are slightly different from the names and descriptions presented in this document. This guide describes a MPEG-4 Decoder core without a built-in memory controller. It instead uses an external ZBT memory controller to store and retrieve motion compensation data.

The MPEG-4 Decoder core receives input samples from a tuner or network interface element that is locking onto a particular channel (or channels), and extracts the bitstreams for the channel(s). That video information is then sent to the MPEG-4 Decoder core, along with a stream number or channel selection.

The control section of the system architecture has the clock generation circuitry and the global reset and start signals. This section also accepts status and error condition codes from the codec and performs related necessary functions.

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

The display interface accepts the decompressed macroblocks from the codec in YUV 4:2:0 8-bit format. An external display controller must change the data format from macroblock processing to a raster type of operation. It must also perform conversion and interpolation of the YUV 4:2:0 signals to the output video format required by the target application (e.g., RGB 4:4:4 format). The MPEG-4 Decoder core interfaces to the controller via the pre-determined format described in Chapter 3, "Basic MPEG-4 Simple Decoder."

The MPEG-4 Decoder core processes a single video frame of information at a time, so that it can seamlessly multiplex between different input data bitstreams. This allows the MPEG-4 Decoder core to operate in a multiple stream environment and minimizes the amount of on-chip memory storage needed for state variables and dynamic register values.

MPEG-4 Simple Decoder Core

The MPEG-4 Simple Decoder core is currently available in 16 different EDIF files and associated wrapper files. The MPEG-4 Simple Decoder core described in this guide does not have built-in memory.

This MPEG-4 Decoder core module can be used to create a larger system configuration by adding a memory controller module, memory, and a display controller interface. A diagram of the additional functional block is presented in Figure 4-2 and can be viewed in the MPEG4_decoder_top.vhd. file located in the following directory.

```
MPEG4_SP_decoder/MPEG4Ver2_decoder/hdl/src
```

This design is used to show the interface to a test bench that will perform a ModelSim simulation of the design. A precompiled core to implement the memory controller has been supplied with the design. It has four read/write channels and can be used with a ZBT SRAM memory.

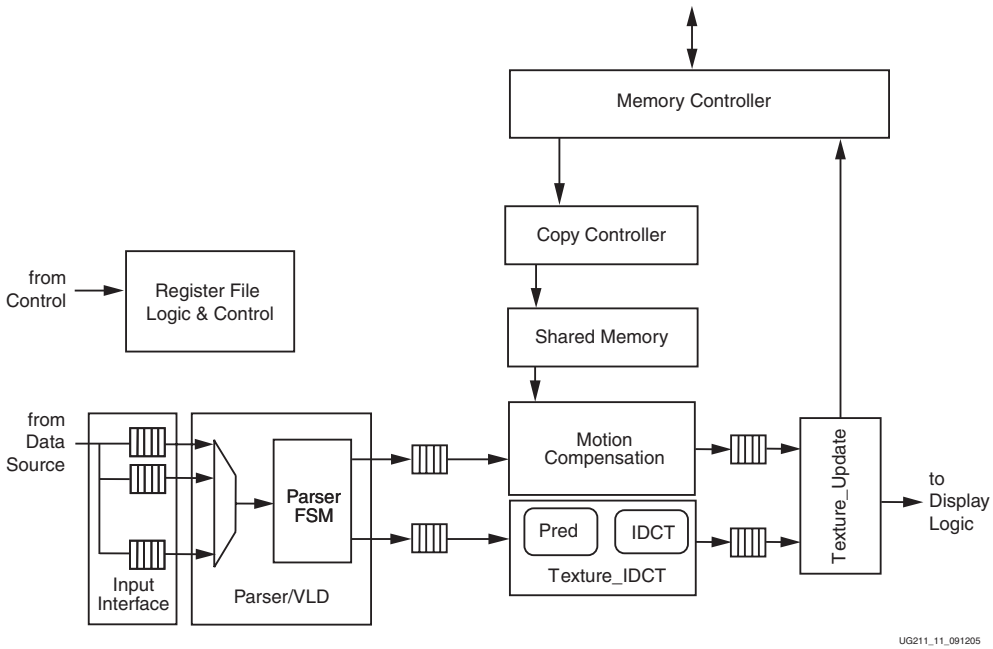

Figure 4-2: **Top Module Block Diagram with Memory Controller**

Display Logic Interface

The uncompressed macroblock data and control data words from the `texture_update` block have been sent to FIFOs located outside of the core. Chapter 5, "Implementing Your Design,"describes how data from this port is stored and used by the simulation program. In a stand-alone system, the data is sent to a display controller which then formats the video information. FIFOs are needed to accumulate a single macroblock of information, since the decoder is a macroblock-based design.

Memory Controller Considerations

The MPEG-4 Decoder core uses one read channel and one write channel of the memory controller. Once 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 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. The `CopyController` module requests a macroblock of data when signaled by the `Parser_VLD` module, and then proceeds to store that macroblock in the local on-chip memory (with a capacity of a little more than a single macroblock 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.

Chapter 5

Implementing Your Design

This chapter describes an architecture that was created in the ModelSim environment to verify overall operation of the MPEG-4 Simple Profile Decoder core in simulation.

Simulation Test Bench

The MPEG-4 Decoder core design package includes a test bench that accesses data from a data file (which represents compressed data of an image sequence). It has the capability to compare the output of the ModelSim-based simulation to stored data files of expected results. The test bench references an MPEG-4 Simple Profile Decoder system design that includes a memory controller, a model for a ZBT SRAM memory, and data checking modules to ensure proper decoder operation.

A block diagram of the supplied test bench is presented in Figure 5-1.

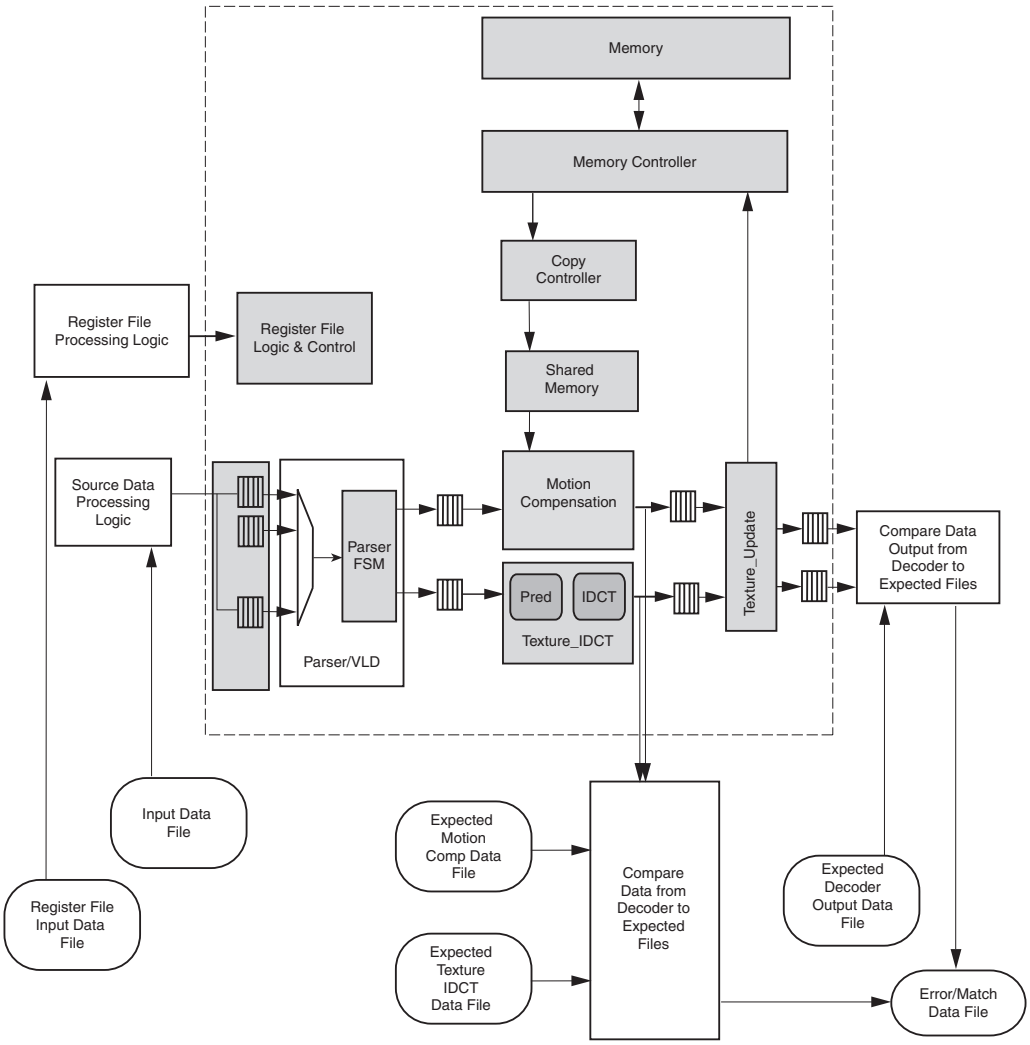

Figure 5-1: **MPEG-4 Simple Decoder Test Bench Simplified Block Diagram**

The test bench file is located in:

MPEG4_SP_Decoder /MPEG4Ver2_Decoder/hdl/simulation/mpeg4_decoder_top_tb.vhd

The corresponding script control file used by ModelSim is

mpeg4_decoder_top_tb.do

and is located in the same test bench directory.

Once ModelSim has been initiated and loaded with the appropriate directories, the test bench program can be loaded, and the DO file executed to display the signals going to and from the MPEG-4 Decoder core.

Once the test bench has been loaded and initiated, it compares the results of the uncompressed video stream as produced by the MPEG-4 codec to the stored results of the C-model generation of the decoder operation within a user-defined acceptable range or DELTA (currently set to 0). The expected files are generated using the batch file (see Appendix B, "Directory Tree Structure") and retrieved from files as defined in the test

bench VHDL code (see Figure 5-2). This figure shows the location of the compressed bitstream input file and the comparison output files.

```
constant TESTCASE_S                 : string := int_2_string(TESTCASE);
type string_array is array (natural range <>) of string(1 to 87);
constant input_file             : string_array(0 to 19) := (
"..\..\testbench\mother_daughter_qcif_30_BR75\data_enc_refer\mother_daughter_qcif_30.bit",
"..\..\testbench\mother_daughter_cif_30_BR200\data_enc_refer\mother_daughter_cif_30.bit ",
"..\..\testbench\foreman_qcif_30_BR200\data_enc_refer\foreman_qcif_30.bit            ",
"..\..\testbench\foreman_cif_5_BR800\data_enc_refer\foreman_cif_30.bit               ",
"..\..\testbench\foreman_cif_10_BR800\data_enc_refer\foreman_cif_30.bit              ",
"..\..\testbench\foreman_cif_30_BR800\data_enc_refer\foreman_cif_30.bit              ",
"..\..\testbench\mobile_qcif_30_BR500\data_enc_refer\mobile_qcif_30.bit              ",
"..\..\testbench\mobile_cif_30_BR3000\data_enc_refer\mobile_cif_30.bit               ",
"..\..\testbench\news_cif_30_BR800\data_enc_refer\news_cif_30.bit                    ",
"..\..\testbench\crew_qcif_15_BR200\data_enc_refer\crew_qcif_15.bit                  ",
"..\..\testbench\crew_cif_30_BR1000\data_enc_refer\crew_cif_30.bit                   ",
"..\..\testbench\crew_vga_30_BR8000\data_enc_refer\crew_vga_30.bit                   ",
"..\..\testbench\crew_4cif_30_BR8000\data_enc_refer\crew_4cif_30.bit                 ",
"..\..\testbench\city_qcif_15_BR200\data_enc_refer\city_qcif_15.bit                              ",
"..\..\testbench\city_cif_30_BR1000\data_enc_refer\city_cif_30.bit                               ",
"..\..\testbench\city_vga_30_BR8000\data_enc_refer\city_vga_30.bit                   ",
"..\..\testbench\city_4cif_30_BR8000\data_enc_refer\city_4cif_30.bit                             ",
"..\..\testbench\harbour_qcif_15_BR100\data_enc_refer\harbour_qcif_15.bit            ",
"..\..\testbench\harbour_cif_30_BR500\data_enc_refer\harbour_cif_30.bit              ",
"..\..\testbench\harbour_4cif_30_BR8000\data_enc_refer\harbour_4cif_30.bit           "
        );
constant mc_exp_file : string := "simulation\testcases\decoder_test" & TESTCASE_S & "\mc_exp.dat";
constant idct_exp_file : string := "simulation\testcases\decoder_test" & TESTCASE_S & "\idct_exp.dat";
constant dec_exp_file : string := "simulation\testcases\decoder_test" & TESTCASE_S &
"\decoder_exp.dat";
constant dec_actual_file : string := "simulation\testcases\decoder_test" & TESTCASE_S &
"\decoder_actual.dat";
constant dec_info_file : string := "simulation\testcases\decoder_test" & TESTCASE_S &
"\decoder_info.txt";
constant texture_update_file : string := "simulation\testcases\decoder_test" & TESTCASE_S &
"\texture_update.txt";
type raw_file is file of character;
file inputFilePtr                   : raw_file open read_mode is input_file(TESTCASE);
file MCExpectedFilePtr              : raw_file open read_mode is mc_exp_file;
file IDCTExpectedFilePtr            : raw_file open read_mode is idct_exp_file;
file DecExpectedFilePtr             : raw_file open read_mode is dec_exp_file;
file DecActualFilePtr               : raw_file open write_mode is dec_actual_file;
file DecInfoFilePtr                 : text open write_mode is dec_info_file;
```

Figure 5-2: **Test Bench Definitions**

The simulation code supplied with this release provides an example of activity diagrams to illustrate the amount of activity that a particular module has during the simulation of the complete module. For example, the amount of time the finite state machine (FSM) of the Parser/VLD module is performing a function (and not waiting for data) can be expressed by a user-defined signal. The activity diagram uses many CPU cycles for the simulation. If a faster simulation is needed, the commands in the simulation script file should be temporarily eliminated.

A directory corresponding to the TESTCASE generic in the test bench (e.g., `simulation/testcases/decoder_test5` for TESTCASE: = 5) must exist and contain the necessary and expected files for the particular image being tested. The image file currently being processed is `foreman_cif_30`. The directory labeled `decoder_test5`

contains files that can be used to perform the simulation. You can create new files in additional directories to perform some custom sequence testing using the available software tools.

Wildcard-II Hardware Verification Platform

The previously described test bench was used to derive an application implemented on a Wildcard-II platform. That application centers around a module that uses the MPEG-4 Decoder core along with some additional modules to interface to the hardware located on the Wildcard-II. The Wildcard-II platform is a PCMCIA-based board that interfaces to a laptop. At the heart of the Wildcard-II is a 2V3000 -4 Xilinx FPGA. The card has a ZBT memory for temporary frame storage. The MPEG-4 Decoder core packed macroblocks in YUV 4:2:0 format will be sent to the PC via a display controller formatter and a LAD bus interface. It is then converted to RGB 4:4:4 format and displayed on the laptop monitor using DirectX. A block diagram of that module is shown in Figure 5-3. The LAD bus interface is a special interface used on the Wildcard-II platform.

The Wildcard-II is a hardware platform that allows an application to incorporate the MPEG-4 Decoder core. A program is written that allows the PC bus to interface to the decoder allowing bi-directional traffic, and demonstrating the complete functionality of the MPEG-4 Decoder core. Currently, the Virtex-II family of chips is represented on the platform with the 2V3000-4 part.

There is a program that allows the MPEG-4 Decoder core to be tested in two different approaches on the hardware platform—processing individual image sequences (the test X selection) or testing multiple sequences (the test Y selection). By selecting the executable file, a menu appears in a DOS window that asks the user to select one of the two options (see "The Wildcard-II C Program Model").

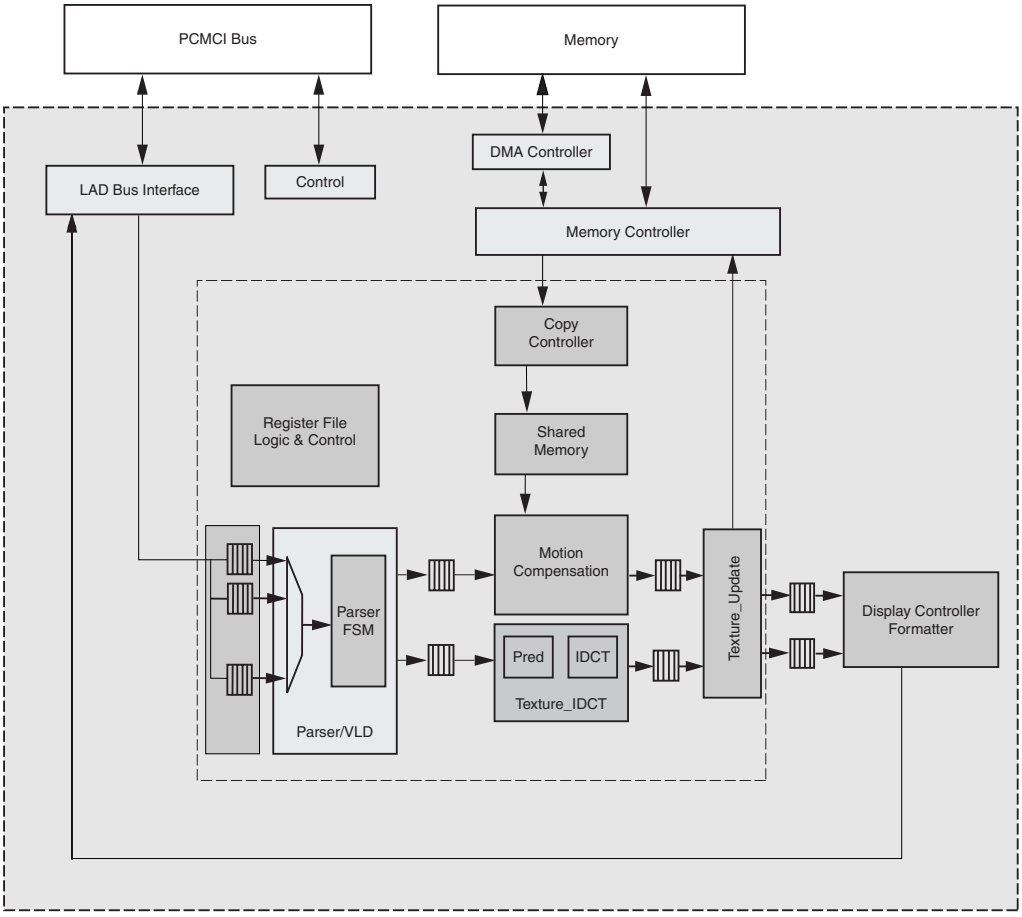

Figure 5-3: **MPEG-4 Simple Decoder Virtual Socket Architecture Block Diagram**

After selecting one of the options, a second menu appears in the DOS window allowing you make a selection from a choice of available images, ranging from QCIF resolution to 4CIF resolution. Since a large amount of data is transferred in both directions over the PCI bus, real-time operation for 4CIF is not achieved. The MPEG-4 Decoder core has been tested and verified in real-time 4CIF on a standalone platform.

Running a Wildcard-II Demonstration

Use the following instructions to run a single-stream decoder hardware demonstration. For information about where to purchase the card, and which drivers have been tested, see Appendix A, "Related Information"

1. Insert a Wildcard-II into a PCMCIA slot.
2. From a Windows Explorer window, double-click on:

   MPEG4_SP_Decoder\WildCard-II\HW_Accel\test_virtual_socket_win32\virtualsocket_ddraw.exe

   A transcript window appears on the desktop with a top-level debug menu.

3. In this window, type X and press Enter.
4. Select a sequence number to test and press Enter.
5. Watch the DirectX video, making note of performance characteristics listed in the transcript window at the end of the run.
6. Repeat step 3 through step 5 (maximum repetitions is 32).
7. To exit, type 'q' at the top-level debug menu and close the window.

Running a Multi-Stream Decoder Hardware Demonstration

Use the following instructions to run a multi-stream decoder hardware demonstration.

Note: This feature will be included in future releases.

1. Insert a Wildcard-II into a PCMCIA slot.
2. From a Windows explorer window, double-click on

   Wildcard -II\HW_Accel\test_virtual_socket_win32\virtualsocket_ddraw.exe

   A transcript window appears on the desktop with a top-level debug menu.
3. In this window, type Y and press Enter.
4. Select the number of sequences (options: 2, 4, or 8) to test and press Enter.
5. Select whether to use the default streams (Y or N) and press Enter.
   a. If you select 'y', then the demo should start automatically.
   b. If you select 'n', then you will be asked one by one which sequences you would like to use.

   Note: the Wildcard-II only has 2 MB of ZBT SRAM on board, which is the equivalent of about three 4CIF frames. Choosing more than this will overwrite and corrupt the reference frames.
6. Watch the DirectX video, making note of performance characteristics listed in the transcript window at the end of the run.
7. Repeat step 3 through step 6 (maximum repetitions is 4).
8. To exit, type 'q' at the top-level debug menu and close window.

The Wildcard-II C Program Model

The executable program as well as the C program workspace resides in the following directory.

`MPEG4_SP_decoder/WildCard-II/HW_ACCEL/test_virtual_socket_win32`

The pre-compiled executables are named virtualsocket_ddraw.exe (no writing of output file) and virtualsocket_ddraw_debug.exe (writes output YUV file). The YUV format is YUV planer with Y followed by U followed by V on a frame-by-frame basis.

Note: You may see a “jerky” video display due to writing of file in background.

The following text is shown in the command window that opens when the executables are run. Note that virtualsocket_ddraw_debug.exe is also used in the regression test. See Appendix C, “Verification and Interoperability.”

```
Please select a test from the Debug menu
   x. Verify MPEG-4 SP Decoder <single stream>
   y. Verify MEGP-4 SP Decoder <multiple streams>
Please select menu choice <q for quit> >>
```

XILINX®

Appendix A

Related Information

This appendix references related useful information and material not otherwise included with MPEG-4 Simple Profile Decoder core documentation. This includes links to web pages for trade groups, specifications, articles, etc.

Documentation

The release also uses and contains code specifically designed by Annapolis Micro Systems that has been supplied with their Wildcard II board.

The data sheet and any applicable application notes or documentation have been supplied in the DOCS directory.

Application Software

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

Table A-1: **Supported Application Software**

Program/Utility	Version	Application
ModelSim	5.8c SE	Simulation Environment
Synplicity Synplify Pro	7.5.1.	VHDL Synthesizer
Xilinx ISE	6.2.03i	Place and Route Tools
Microsoft Visual Studio C++	6.0	Software Development Platform
ActivePerl	5.8.3	Interpreter used for regression test (http://www.activestate.com/Products/ActivePerl/)
Annapolis Micro Systems WildCard-II Platform	API: 2.6 Driver: 3.3 Firmware: 1.6	Hardware Verification Platform (http://www.annapmicro.com/)

Supported Hardware Platform

A useful addition to this core is the Wildcard-II board from Annapolis Microsystems, Inc. Annapolis has boards that can be inserted into a laptop computer and demonstrate the

MPEG-4 Simple Decoder 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 2V3000 -4 FPGA, ZBT memory, and an interface to the PCMCI bus.

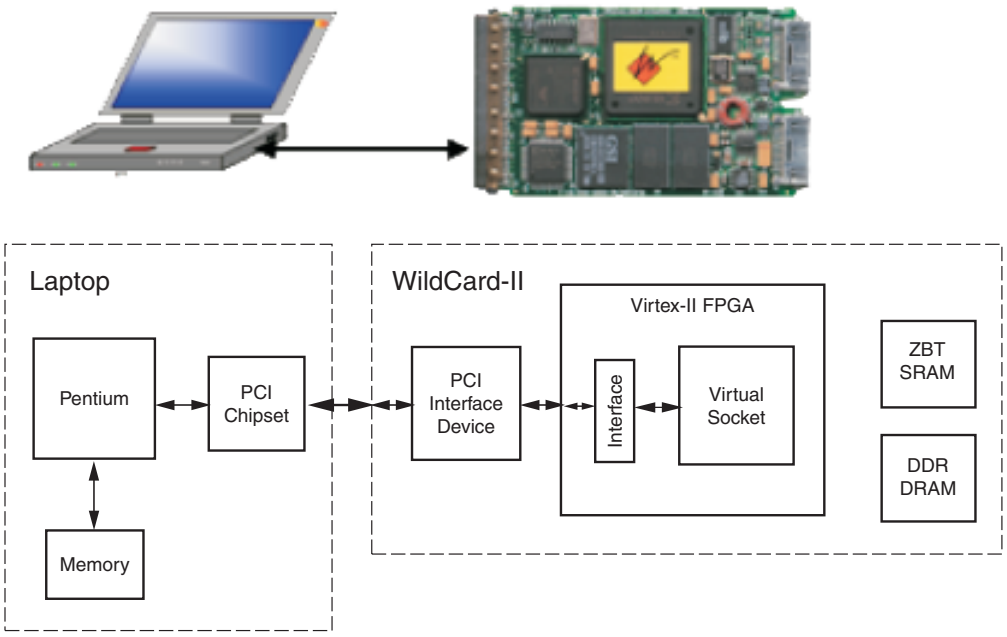

Figure A-1: **Wildcard-II Hardware Verification Platform, Block Diagram**

To demonstrate the MPEG-4 Simple Decoder, Xilinx designed a software-based system that interfaces to this hardware platform under the control of the microprocessor on the laptop. There is a pre-compiled executable that controls a variety of test and display features to verify the decoder 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 2V3000 located on the hardware platform. That bit file houses a number of modules along with the MPEG-4 Simple Decoder that performs image restoration from the compressed bitstream file presented to it. The decompressed image is then passed back to the PC processor where it is then displayed on the monitor, and subjective evaluations can be performed.

Library and Test Bench Directory Creation

Located inside the directory supplied with the core under the decoder HDL branch are either one or two batch files (depending on the type of release namely with or without source code) that will create all necessary libraries and test bench directories needed for the ModelSim simulation environment. The normal presentation of the core will be without decoder specific source code.

These two files are called CompileLibraries.bat and CreateTBfiles.bat. These files call a number of TCL files that create the needed structure in the

MPEG4_SP_decoder /XLIB/libraries and the

MPEG4_SP_decoder /MPEG4Ver2_decoder/hdl/simulation/

test cases directories. There can be twenty directories made in the test cases directory.

If source code is not provided, a precompiled ModelSim library environment using ModelSim version 5.8 will be provided and located in the XLIB directory located near the top of the directory structure. The CompileLibraries.bat file will not be provided for this case since the user will not be able to compile the libraries. The user will have to use ModelSim version 5.8.c since that was used to create the libraries. Different versions of the simulator will need the libraries to be generated with the identical version.

After executing the CreateTBfiles.bat batch file, a menu will appear that allows the user to select which image file will be the source for the current ModelSim operation. These test case numbers correspond to the TESTCASE generic used in the test bench file,

MPEG4_SP_decoder/MPEG4Ver2_decoder/HDL/simulation/mpeg4_decoder_top_tb.vhd.

Therefore, the only files that will be created are the ones needed for the current environment, therefore saving significant disk space. (Note that creating the test bench files for all twenty streams will consume over 3 GB of disk space.)

The following is the menu presented after executing the test bench batch file. To run the simulation files supplied in the release, sequence 5 must be selected so that the needed files will be created. It is the default condition in the TESTCASE generic.

```
Testbench File Creation Menu
Note: sequence 5: Foreman CIF used in TESTCASE generic
0: Mother/daughter QCIF
1: Mother/daughter CIF
2: Foreman QCIF
3: Foreman CIF < 5 fps>
4: Foreman CIF <10 fps>
5: Foreman CIF <30 fps>
6: Mobile QCIF
7: Mobile CIF
8: News CIF
9: Crew QCIF
a: Crew CIF
b: Crew VGA
c: Crew 4CIF
d: City QCIF
e: City CIF
f: City VGA
g: City 4CIF
h: Harbour QCIF
i: Harbour CIF
j: Harbour 4CIF
q: Quit
Select menu choice:
```

In order for these files to operate correctly, the user path must point to the location of the vsim command (for example, `C:/ModelSim_5.8c/win32`) and the default `modelsim.ini` file must contain the xilinxcorelib directory that points to the location of the library. An example of the library portion of the `modelsim.ini` file is presented below. The user is responsible to set the appropriate parameters in his control files to execute the simulations programs. Each user environment will be unique.

```
[Library]
std = $MODEL_TECH/../std
ieee = $MODEL_TECH/../ieee
verilog = $MODEL_TECH/../verilog
vital2000 = $MODEL_TECH/../vital2000
std_developerskit = $MODEL_TECH/../std_developerskit
synopsys = $MODEL_TECH/../synopsys
modelsim_lib = $MODEL_TECH/../modelsim_lib
SIMPRIM = C:\Xilinx\vhdl\mti_se\simprim
UNISIM = C:\Xilinx\vhdl\mti_se\unisim
XILINXCORELIB = C:/Xilinx/vhdl/mti_se/XilinxCoreLib
```

XILINX®

Appendix B

Directory Tree Structure

The following diagrams provide information about the structure of the MPEG-4 Simple Profile Decoder files and directories. Files that have been referred to in other sections of this guide are included in the structure presented in this Appendix.

Figure B-1 shows the entire tree structure of the MPEG-4 Simple Decoder core release starting at the top level with many of the sub-directories.

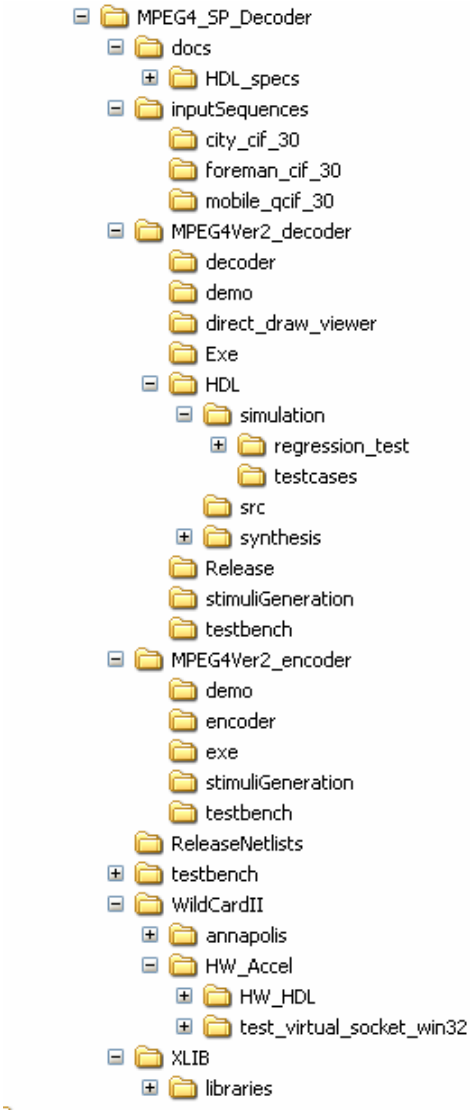

Figure B-1: **MPEG-4 Decoder Directory Structure**

Figure B-2 shows the test bench directory and the names of the different sequences that have data files associated with each sequence,

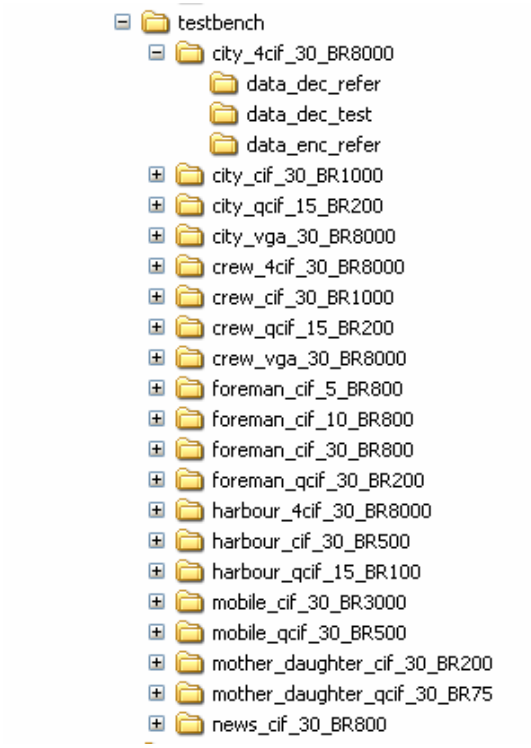

Figure B-2: **Test Bench Subdirectory and Folders**

Figure B-3 shows the layout of the XLIB library structure used by ModelSim.

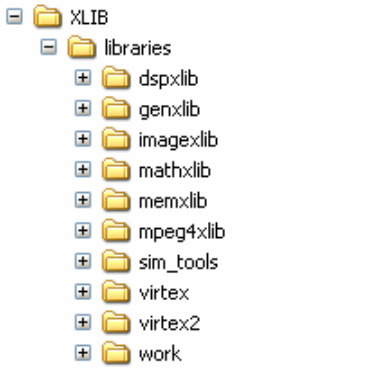

Figure B-3: **XLIB Library Folders (after Compilation)**

XILINX®

Appendix C

Verification and Interoperability

Regression Testing

To ensure that the MPEG-4 Simple Decoder module is performing properly, the current release includes a C program workspace and an executable file that will verify the operation of the MPEG-4 Simple Decoder core codec. The software will reference a pre-defined number (20) of stored image sequences ranging from QCIF to 4CIF resolutions and compare those files to ones produced by the MPEG-4 core implemented on the Wildcard-II hardware platform. The regression test will determine if the expected results match the output of the core and identify the number of errors that occur for each image sequence.

To perform the regression test, the set of images that are 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_decoder.bat that calls a Perl script file named run_decoder.pl that will create the twenty sequences in the appropriate directory. The batch file is located in the following directory.

```
MEPG4_SP_decoder_rel1_0 /testbench
```

The generated image files will be stored in the following directory where the particular image sequence name has been predefined.

```
MEPG4_SP_decoder_rel1_0 /testbench/{particular image sequence
name}/data_dec_refer
```

The regression test software will place the newly reconstructed YUV files derived using the Wildcard-II in the adjacent *data_dec_test* directory.

The regression test C model workspace resides in the following directory and is named regression_test.dsw.

```
MEPG4_SP_decoder_rel1_0
/MPEG4_decoder/hdl/simulation/regression_test
```

Upon execution of the regression test executable, a series of questions are displayed for the user to answer as seen below.

```
Do you want to re-build the EDIF file in Synplicity?

Do you want to re-build the X86 file for test x?

Do you want to re-build the Reference streams?

Do you want to run test x on the Wildcard-II?

A simple yes (y) or no (n) answer will suffice.
```

The answer to the first two questions is no (n) because you do not have to rebuild any of the files, since they are provided. If the image sequences have been generated prior to the execution of the test, they do not need to be regenerated. If they have not been generated,

they must now be regenerated. Therefore, the answer to the third question is yes (y). It is recommended that these files be generated using the aforementioned batch file. The answer to the last question will be yes (y), and will perform the reconstruction of all image sequences using the Wildcard-II platform.

The test completes with information concerning the number of errors found by the testing process for each individual sequence.

If you select no for each of the four questions listed above, then the test will use any files already stored in the testbench directory to perform the checking process.

List of Sequences in the Regression Test

The following table contains the list of sequences used in the regression test.

Table C-1: **Regression Test Sequences**

Sequence	Resolution	Target Bit Rate (kbps)	Frames/sec
Mother_daughter	QCIF	75	30
Mother_daughter	CIF	200	30
foreman	QCIF	200	30
foreman	CIF	800	5
foreman	CIF	800	10
foreman	CIF	800	30
mobile	QCIF	500	30
mobile	CIF	3000	30
news	CIF	800	30
crew	QCIF	200	15
crew	CIF	1000	30
crew	VGA	8000	30
crew	4CIF	8000	30
city	QCIF	200	15
city	CIF	1000	30
city	VGA	8000	30
city	4CIF	8000	30
harbour	QCIF	100	15
harbour	CIF	500	30
harbour	4CIF	8000	30

XILINX®

Appendix D

Using the MPEG-4 SP Decoder and Encoder C-Models

Overview

This appendix provides instructions for generating a software demonstration using the C-model in both Decoder and Encoder modes.

MPEG-4 Simple Decoder C-Model

In the process of designing the decoder in VHDL, a C based software decoder model was built to help verify proper operation of the overall system design as well as specific sub-section functionality. The software accepts compressed image bitstreams selected by the Microsoft C Studio environment. A number of image sequences as well as the compressed bitstream of a variety of images with different sets of parameters have been stored to be viewed by a viewing application. These files are located in the same directory mentioned in Appendix C, "Verification and Interoperability."

The C model program for the MPEG-4 Simple Decoder uses the following directories.

```
MEPG4_SP_Decoder_rel1_0/MPEG4Ver2_decoder

MEPG4_SP_Decoder_rel1_0/MPEG4Ver2_decoder/decoder

MEPG4_SP_Decoder_rel1_0/MPEG4Ver2_decoder/debug

MEPG4_SP_Decoder_rel1_0/MPEG4Ver2_decoder/demo

MEPG4_SP_Decoder_rel1_0/MPEG4Ver2_decoder/direct_draw_viewer

MEPG4_SP_Decoder_rel1_0/MPEG4Ver2_decoder/exe
```

The workspace file is located in the first directory, the MPEG4_decoder directory (for short) and is called MEPG4_SP_Decoder_rel1_0.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 Simple Decoder core. This file contains important defines that dictate the operation of the C code. These defines are located in the AllInOne.h file and steer the C program results, and are shown in the following example.

Two important directives in the following list are the WRITE_FILE and the WRITE_TB_FILES.

```
//#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 WRITE_TB_FILES //Uncomment to create HW test bench files
#define VIEW_STATISTICS//important statistical information

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

When expected files are being created for the test bench, the WRITE_TB_FILES statement has to be defined. When the regression test is being performed the WRITE_FILE statement has to be defined. The code has been designed to allow a variety of operations for the user to perform. Therefore, there will be executable code already pre-defined for operations needed for this release, such as the executable that creates the image sequence test stream for the regression test. These executables will be located in the exe directory defined above. The decoder directory contains source code for the C model where the other directories contain additional support files. Note that all necessary executables will be provided in the exe directory and the user can use the software at his/her own risk. Any changes to the C code will require the Visual C software and license.

Running the Decoder C-Model

This section contains instructions for generating a compressed bitstream from MPEG-4 SP Decoder C-Model software.

Running the Software Demonstration Using the Default Stream

1. Open a Windows Explorer window.
2. Double-click on the following path:

   `\MPEG4_SP_Decoder\MPEG4Ver2_Decoder\exe\MPEG4Ver2_decoder_ddraw.exe`

   A transcript window opens on the desktop. The default stream that is used is Foreman CIF.

Running The Software Demonstration Using a Non-default Stream

1. *Using Visual Studio.*
   a. Double-click on the following path to launch a Visual Studio Window:

      `\MPEG4_SP_decoder\MPEG4Ver2_decoder\MPEG4Ver2_decoder.dsw`

   b. Specify the control file by selecting Project > Settings.
   c. Click the Debug tab and set the program arguments.
   d. Generate the control file name if one does not already exist.
   e. Compile and execute the code.
2. *Using a DOS command.*
   a. Open a DOS window.

b. Change directory to:

`\MPEG4_SP_decoder\MPEG4Ver2_decoder\Release`

c. At the DOS prompt, type the following:

`MPEG4_SP_decoder_ddraw.exe<ctl_file>`

where `<ctl_file>` is the name of the desired control file.

MPEG-4 Encoder C-Model

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

Running the Encoder C-Model

This section contains instructions for generating an MPEG-4 SP Encoder C-model software demonstration.

To run the Encoder C-Model software

1. Ensure that the forman_cif_30.yuv file is in the following directory.

   `MPEG4_SP_decoder\InputSequences\foreman_cif_30`

2. Ensure that the foreman_cif_30.cfg file is in the following directory.

   `MPEG4_SP_decoder\testbench\foreman_cif_30_BR800`

   After executing the MPEG4Ver2_encoder.exe file in the MPEG4_SP_decoder\MPEG4Ver2_encoder\Exe directory, the following transcript will appear. After a short period of time the file will be generated and located in the appropriate test bench directory.

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.