Department of
CAMSI - Conception d'Architectures de Machines et de Systèmes Informatiques

MASTER THESIS

"Video decoding: SDI interface implementation & H.264/AVC bitstream decoder hardware architecture design and implementation"

Author:

Mr. Vicheka PHOR

School supervisor:

Mr. Jacques JORDA

Company advisor:

Mr. Phillip WEISSFLOCH

Internship duration:

5 months and a half

29 August 2014

Academic Year 2013-2014

Department of
CAMSI - Conception d'Architectures de Machines et de Systèmes Informatiques

MASTER THESIS

"Video decoding: SDI interface implementation & H.264/AVC bitstream decoder hardware architecture design and implementation"

Author:

Mr. Vicheka PHOR

School supervisor:

Mr. Jacques JORDA

Company advisor:

Mr. Phillip WEISSFLOCH

Internship duration:

5 months and a half

29 August 2014

Academic Year 2013-2014

CONTENTS

Chapter III: H.264/AVC bitstream decoder hardware design in Verilog models

LISTS OF FIGURE

LISTS OF TABLE

ABBREVIATIONS

3G	Third Generation
AVC	Advance Video Coding
CABAC	Context-based Adaptive Binary Arithmetic Coding
CABAD	Context-based Adaptive Binary Arithmetic decoding
CAVLC	Context-based Adaptive Variable Length Coding
CAVLD	Context-based Adaptive Variable Length decoding
CBR	Constant Bit Rate
CPB	Coded Picture Buffer
DCT	Discrete Cosine Transform
DDR	Double Data Rate
DPB	Decoded Picture Buffer
DUT	Decoder under test
DVI	Digital Visual Interface
FIFO	First-In, First-Out
FLC	Fixed Length Coding
FPGA	Field Programmable Gate Array
HD	High Definition
HDL	Hardware Description Language
HDMI	High Definition Multimedia Interface
HRD	Hypothetical Reference Decoder
HSS	Hypothetical Stream Scheduler
IDR	Instantaneous Decoding Refresh
IP	Intellectual Property
IT/IQ	Inverse Transform / Inverse Quantization
ITU-T	International Telecommunication Union - Telecommunication standardization sector
JM	Joint Model
JTAG	Joint Test Action Group
LDR	Low Data-Rate
LN	Line Number
LSB	Least Significant Bit
LUT4	4-input Look Up Table
MB	Macroblock
MBAFF	Macroblock-Adaptive Frame-Field Coding
MPEG	Moving Picture Experts Group
MSB	Most Significant Bit
MVC	Multiview Video Coding
NAL	Network Abstraction Layer
PCI	Peripheral Component Interconnect
PCS	Physical Coding Sublayer
PHY	Physical Layer
PLL	Phase Lock Loop
QP	Quantization Parameter
RAM	Read Access Memory
RBSP	Raw Byte Sequence Payload
R&D	Research and Development
ROM	Read Only Memory
RX	Receiver
SD	Standard Definition

SDI	Serial Digital Interface
SDK	Software Development Kit
SERDES	Serializer/Deserializer
SEI	Supplemental Enhancement Information
SMPTE	Society of Motion Picture and Television Engineers
SODB	String Of Data Bits
SVC	Scalable Video Coding
TMDS	Transition Minimized Differential Signaling
TX	Transmitter
USB	Universal Serial Bus
UUID	Universal Unique Identifier
VBR	Variable Bit Rate
VCL	Video Coding Layer
VLC	Variable Length Coding
VPB	Video Protocol Board
VUI	Video Usability Information
XAUI	Xilinx 10 Gigabit Attachment Unit Interface

ACKNOWLEDGEMENT

Every project big or small is successful largely due to the effort of a number of great people who have always given their valuable advice or lent a helping hand. I sincerely appreciate the inspiration, support and guidance of all those people.

I, Mr. Vicheka PHOR, the student of University Toulouse III - Paul Sabatier, am extremely grateful to Enciris Technologies Company, Mr. Phillip WEISSFLOCH and Ms. Cornelia WEISSFLOCH, the director and the co-director of the company, for giving me an opportunity to undergo my internship.

At this juncture, I feel deeply honored in expressing my sincere thanks to Mr. Malik CISSÉ, Mr. Rémi DUPIN, Mr. Fréderic REQUIN, and Mr. Mickaël POSTOLOVIC, who are engineers at the company for their guidance, their kindness, their time and their help during my internship.

I would like also to thank to all staffs at the company for their generous attitudes and friendly behavior.

Also, I would like to thank Mr. Abdelaziz M'ZOUGHI and Mr. Jacques JORDA, the directors of CAMSI department for accepting me to study the Master 2 professional degree in CAMSI department.

I would like also to express my gratitude to Mr. François THIEBOLT, a professor in CAMSI department for his support, his time and his encouragement.

I would like also to show my appreciation to Mr. Ponia PECH, who is an engineer at company and who is like a brother to me for his kind-heart, his support, his inspiration and his great help.

In the end, I would like to place a deep sense of gratitude to my family members and my friends who have been constant source of inspiration during my internship.

ABOUT ENCIRIS TECHNOLOGIES COMPANY

Enciris Technologies Company, which was founded in February 2006 in Gaillac, France, is an internationally oriented company, a leading designer and manufacturer of high performance video processing hardware for OEMs (Original Equipment Manufacturer) medical video, internet broadcasting, security and surveillance, inspection systems and robotics, and professional users. It provides solutions for customers aiming to add affordable HD video compression, streaming, and storage to their products or services.

Specialized in high definition video processing, Enciris Technologies has developed its own HDTV compression and decompression technology that complies with widely used standards such as VC-1/SMPTE-421M and H.264. Enciris Technologies is Lattice LEADER design services partners. Its activities include:

- design and manufacture high definition video systems,
- design and develop board and module level products incorporating Lattice FPGAs,
- develop IP for FPGAs,
- build VC-1/SMPTE-421M and H.264 solutions.

Enciris Technologies can modify its own standard products or design hardware from scratch to meet your video acquisition, compression, or processing requirements.

Contact

Company:	Enciris Technologies, SAS
Number and street:	22, Avenue de l'Europe
City:	Gaillac
Postal code/zip:	81600
Region:	Midi-Pyrénées
Country:	France
Telephone:	+33 581 180 112
Fax:	+33 826 420 835
Email:	info@enciris.com

Other information

Launched:	February 2006
Number of employees:	9
Sales (mil):	$1.11

INTRODUCTION

Nowadays, digital video is widely used in many applications in the way of creating or sharing for example the digital television broadcasting, internet video streaming, mobile video streaming, DVD video and video calling application. Therefore, effective video coding is an essential component of these applications and can make the difference between the success and failure of a business model. The video coding is the process of compressing and decompressing a digital video signal which is sent/received by various interfaces such as: SD-SDI, HD-SDI, 3G-SDI, DVI and HDMI. There are many methods or algorithms to compress and decompress digital video such as: VC1, H.262/MPEG-2, H.263/MPEG-4, and H.264/AVC algorithm. Among these video coding algorithms, H.264/AVC has huge significance to the broadcast, internet, consumer electronics, mobile and security industries application. Also, H.264/AVC is the latest in a series of standards published by the ITU and ISO. It describes and defines a method of coding video that can give better performance than any of the preceding standards. H.264/AVC makes it possible to compress video into a smaller space, which means that a compressed video clip takes up less transmission bandwidth and/or less storage space compared to older codecs.

This master thesis is organized in two parts:

- The first part which is composed of one chapter: Chapter I, will study about the SDI video interface by demonstrating the implementation of a Lattice tri-rate SDI PHY IP core on a Lattice FPGA of the Enciris LT-125 board in order to provide an SMPTE 3G/HD/SD SDI video input functionality on the board without using any external chipset like the Gennum chipset which is used today on Enciris boards.

    - Chapter I: Implementation of Lattice tri-rate serial digital interface PHY IP core, will provide an overview of LatticeECP3 SERDES/PCS, an overview of Lattice tri-rate SDI PHY IP core, an overview of LT-125 Enciris board and the implementation of the tri-rate SDI PHY IP core on the Enciris LT-125 board.

- The second part which contains two chapters: Chapter II and Chapter III, will describe about the H.264/AVC video decoding algorithm, particularly the H.264/AVC bitstream decoder hardware architecture design and implementation.

    - Chapter II: H.264/AVC bitstream decoder, will introduce an overview of video processing, an overview of H.264/AVC decoder, H.264/AVC parameters - profile and level, and H.264/AVC bitstream decoder.

    - Chapter III: H.264/AVC bitstream decoder hardware design in Verilog models, will cover the bitstream decoder hardware architecture, tools used in the FPGA development flow design, hardware implementation and results.

PART I

CHAPTER I
"IMPLEMENTATION OF LATTICE TRI-RATE SERIAL DIGITAL INTERFACE IP CORE"

CHAPTER I

"IMPLEMENTATION OF LATTICE TRI-RATE SERIAL DIGITAL INTERFACE PHY IP CORE"

This chapter will demonstrate the implementation of a Lattice tri-rate SDI PHY IP core on a Lattice FPGA of the Enciris LT-125 board in order to provide an SMPTE 3G/HD/SD SDI video input functionality on the board without using any external chipset like the Gennum chipset which is used today on Enciris boards.

Serial Digital Interface (SDI) is the most popular raw video connectivity standard used in television broadcast studios and video production facilities. With SDI, the high resolution video stream can be transmitted through a 75-Ohm coaxial cable for as long as several hundreds meters.

In order to carry out this task, one needs to have the following knowledge:

1. overview of LatticeECP3 SERDES/PCS
2. overview of Lattice tri-rate SDI PHY IP core
3. overview of LT-125 Enciris board
4. implementation of the tri-rate SDI PHY IP core on the Enciris LT-125 board

I.1 Overview of Lattice ECP3 SERDES/PCS

The Lattice ECP3 FPGA family combines a high-performance FPGA fabric, high-performance I/Os and up to 16 channels of embedded SERDES with associated Physical Coding Sublayer (PCS) logic. The PCS logic can be configured to support numerous industry-standard, high-speed serial data transfer protocols such as PCI Express, Gigabit Ethernet (1GbE and SGMII), XAUI plus multiple other standards, and user-specified generic 8b10b mode.

Each channel of PCS logic contains dedicated transmit and receive SERDES for high-speed, full-duplex serial data transfer at data rates up to 3.2 Gbps. The PCS logic in each channel can be configured to support an array of popular data protocols including SD-SDI, HD-SDI and 3G-SDI.

Lattice ECP3 FPGA devices have from one to four quads of embedded SERDES/PCS logic. Each quad, in turn, supports four independent full-duplex data channels (RX and TX). A single channel can support a data link and each quad can support up to four channels with both RX and TX circuits, and an auxiliary channel that contains the TX PLL. The quad SERDES/PCS macro performs the serialization and de-serialization function for four lanes of data.

Figure I.1 describes a simplified SERDES/PCS quad.

More detailed information about LatticeECP3 SERDES/PCS is provided in reference [1].

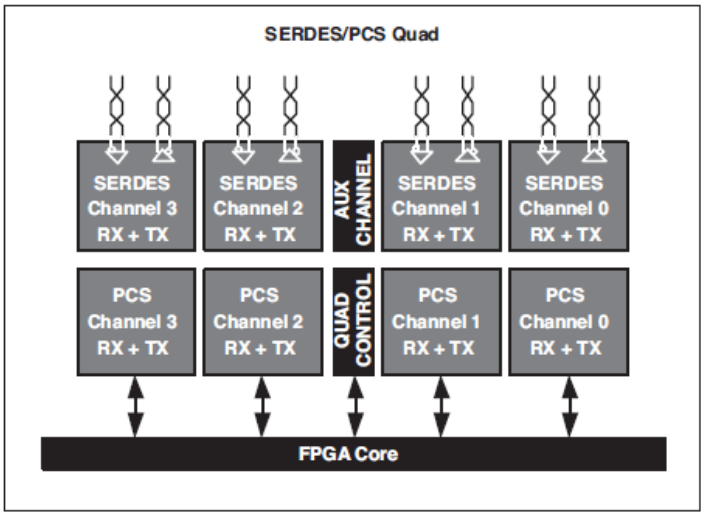


Figure I.1: SERDES/PCS Quad Block Diagram

I.2 Overview of Lattice tri-rate SDI PHY IP core

The Lattice Tri-Rate SDI (Serial Digital Interface) PHY (Physical Layer) IP (Intellectual Property) core is a complete SDI PHY interface that connects to the high-speed SDI serial data on one side (through LatticeECP3™ SERDES) and the formatted parallel video data on the other side. For More detailed information of this IP core is provided in reference [2].

I.2.1 Lattice tri-rate SDI PHY IP core functional description

Lattice's tri-rate SDI PHY IP core consists of the following major functional blocks: SDI encoder/decoder, word alignment, CRC detection and checking, VPID (video payload identifier) insertion and extraction, and rate detection logic. A block diagram of the SDI IP core is given hereafter:

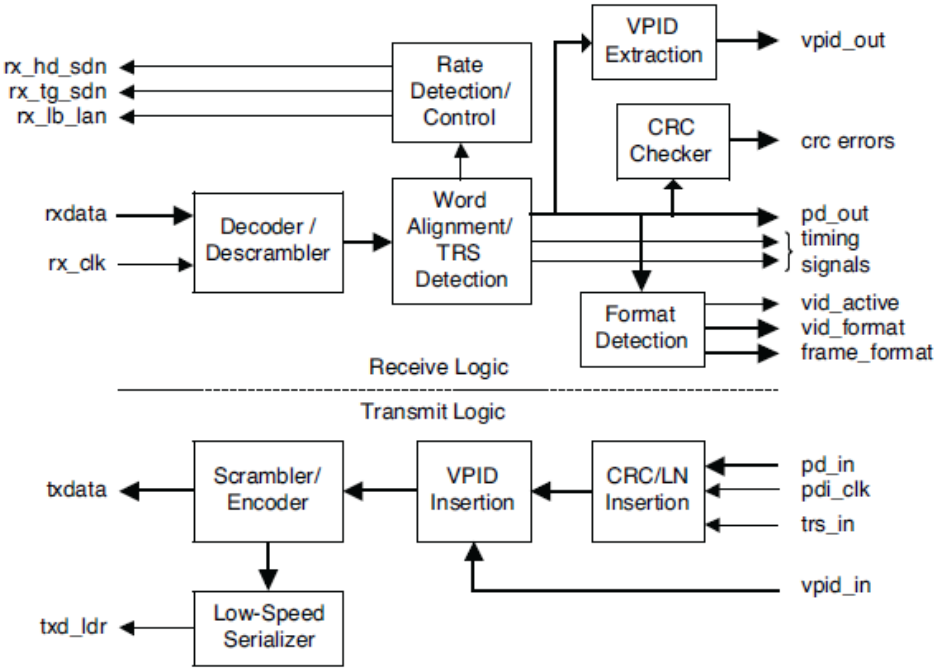


Figure I.2: Tri-Rate SDI PHY IP Core, High-Level Functional Diagram

For a full description of each functional block, please refer to reference [2].

I.2.2 Lattice tri-rate SDI PHY IP core features and applications

The Lattice tri-rate SDI PHY IP core features are:

- Dynamic reception of multiple interface standards over the same physical cable: 270 Mbps SD-SDI, 1.485 Gbps HD-SDI and 2.97 Gbps 3G-SDI interfaces
- Automatic Rx (receive) rate detection and dynamic Tx (transmit) rate selection
- Multiple SD source formats support: SMPTE 125M and SMPTE 267M (13.5 MHz only)
- Multiple HD source formats support: SMPTE 260M, SMPTE 274M, SMPTE 295M and SMPTE 296M
- Support for 3G source formats, including 3G Level-B format: SMPTE 425M
- Word alignment and timing reference sequence (TRS) detection
- Field, vertical blanking (vblank) and horizontal blanking (hblank) timing signals generation
- CRC computation, error checking and insertion for HD/3G
- Line number (LN) decoding and encoding for HD/3G
- Custom source format support for HD/3G
- Video Payload Identifier (VPID) insertion and extraction for HD/3G
- 10-bit parallel input/output support for SD
- Soft-logic based low data-rate (LDR) serializer for SD transmission.

This IP core enables faster development of applications for processing, storing, and bridging SDI video data.

I.2.3 Tri-Rate SDI PHY IP pass-through demo

Above all, it is necessary to test the Lattice SDI IP core feature by implementing the pass-through demonstration scheme on the Lattice evaluation board.

The Tri-Rate SDI PHY IP pass-through is demonstrated on the LatticeECP3 Video Protocol Board (VPB) which is an evaluation board that has the LFE3-95E-7FN1156C FPGA on it. The pass-through design is set up to receive video from a standard SDI source and re-transmit it through the IP to a SDI monitor. The pass-through scheme is shown in Figure I.3.

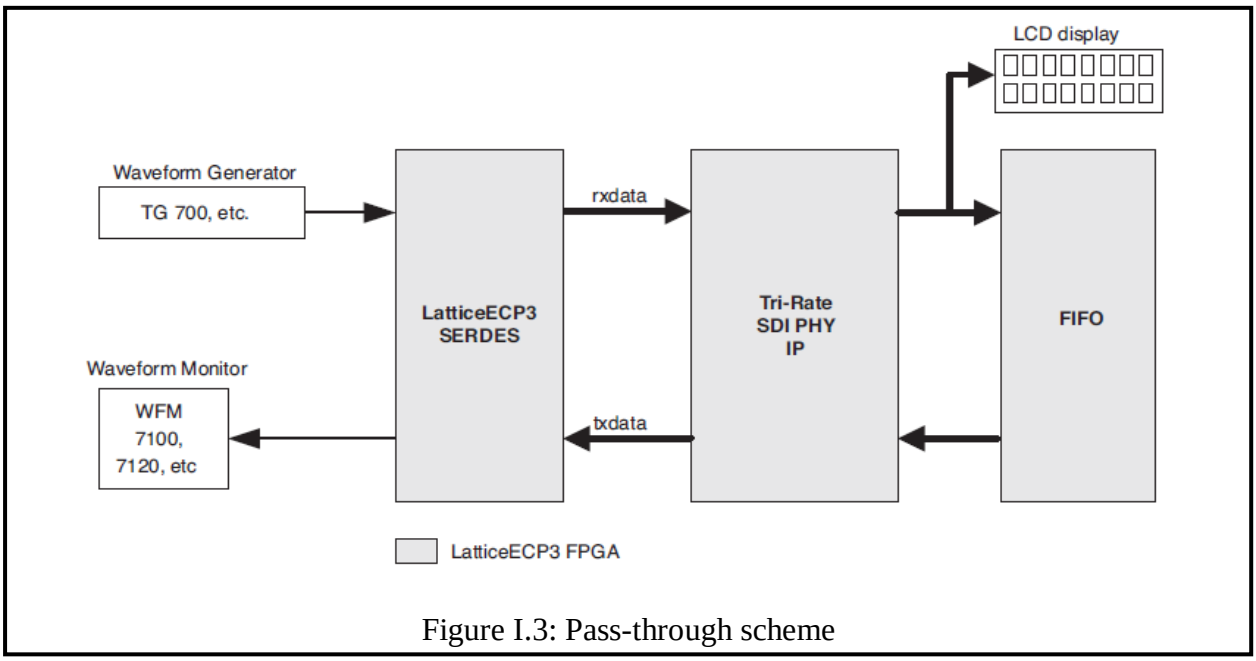


Figure I.3: Pass-through scheme

The bitstream is uploaded on the FPGA device via JTAG connector. The JTAG connector is shown in Table I.1. The Pin 4, Pin 5, Pin 9 and Pin 10 of JTAG connector are left open in order to program the FPGA device.

The Tri-Rate SDI PHY IP pass-through sample design is shown in Figure I.4.

Table I.1: JTAG connector

Pin #	Description
Pin 1	VCC
Pin 2	TDO
Pin 3	TDI
Pin 4	PROGRAMN
Pin 5	NC
Pin 6	TMS
Pin 7	GND
Pin 8	TCK
Pin 9	DONE
Pin 10	INITN

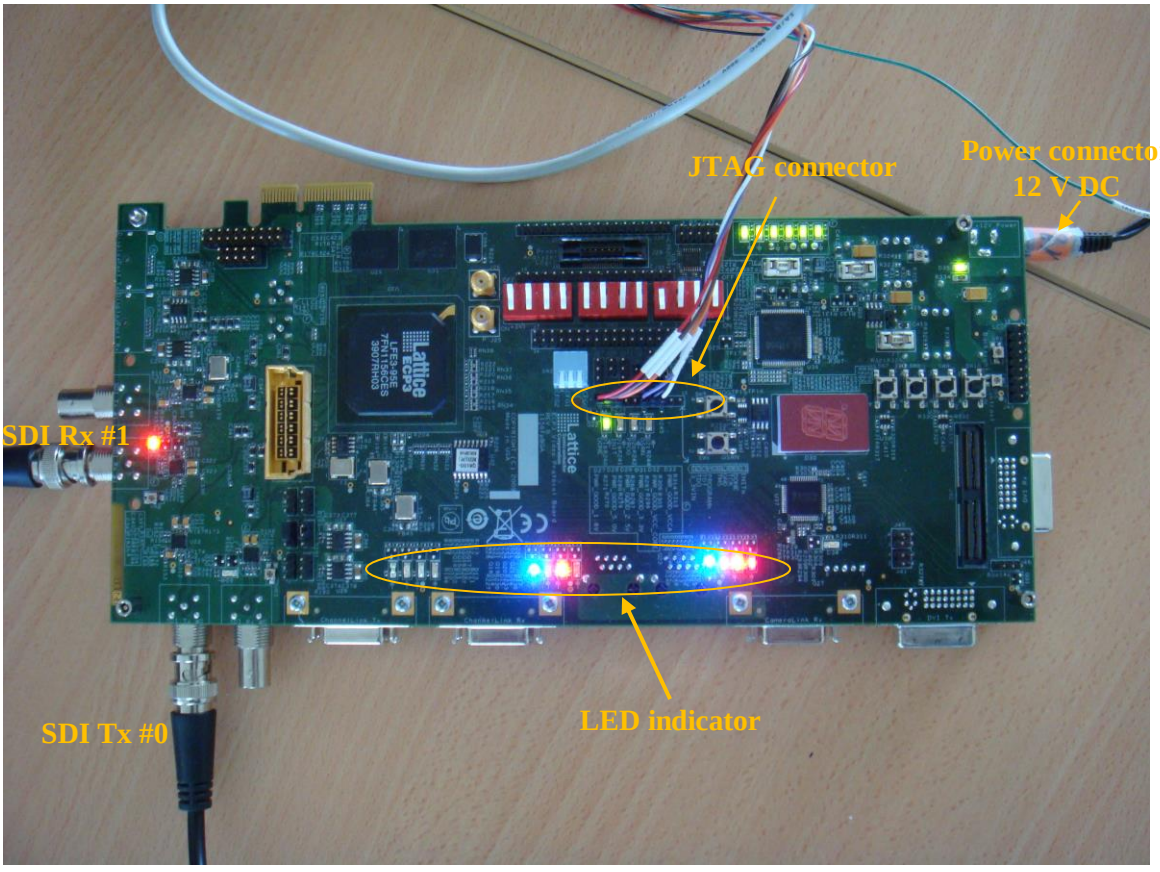


Figure I.4: Tri-Rate SDI PHY IP pass-through sample design

Notes:

- The Tri-Rate SDI PHY IP pass-through demo uses two data channels of SERDES quad A: Channel0 and Channel1. Each data channel is configured for both RX and TX. However, the Channel0 is used for TX only and the Channel1 is used for RX.

- Without IP core license, the IP core will be operational for approximately four hours after initialization. After four hours, the device will stop working and it will be necessary to reprogram the device to re-enable the operation.
- LED indicator displays the status of the SERDES and the video input.

I.3 Overview of Enciris LT-125 board

The LT-125 is a Lattice Semiconductor ECP3 FPGA based board for the evaluation of Enciris Technologies' HD and SD video compression and decompression IP. The LT-125 is designed to demonstrate high performance H.264/AVC and VC-1 encoding and decoding in FPGA applications. This board captures and compresses video from digital HD video sources via DVI, 3G/HD/SDI, HDMI connectors up to a video resolution of 1920x1080@60fps. Simultaneous dual channel video capture is also possible up to a video resolution of 1920x1080@30fps per channel. VC-1/SMPTE-421M is compressed at the advanced profile up to the level 3. H.264 is compressed at the baseline profile up to the level 4.1. An HDMI bypass output is also available.

For more detailed information about this board, see references [3] and [4].

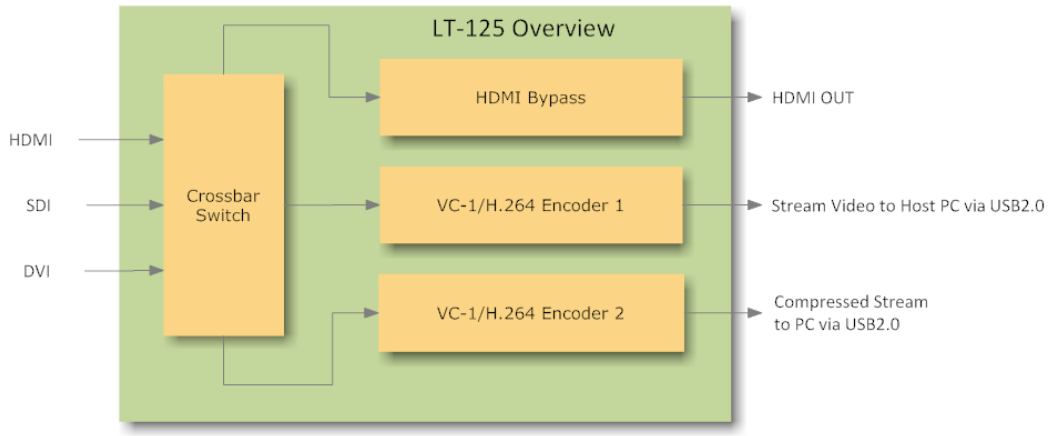


Figure I.5: LT-125 Overview

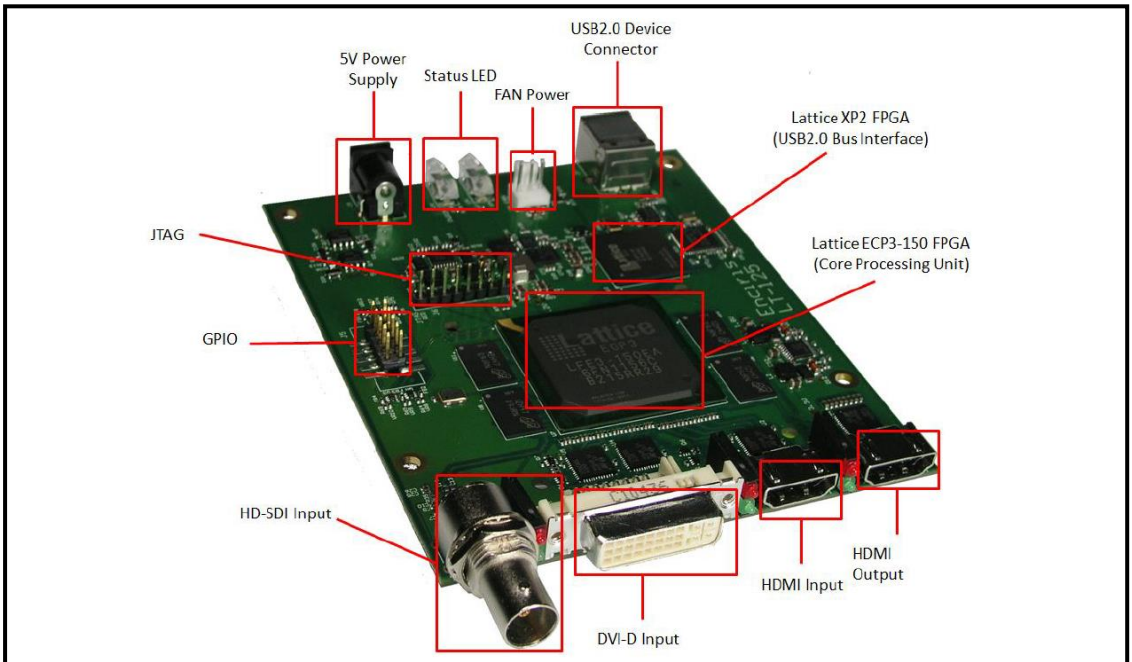


Figure I.6: Top view of the LT-125 Board

I.3.1 LT-125 board's functional description

The LT-125 evaluation board includes video input and output connectors, a series of video input equalizers and a level shifter, a Lattice ECP3-150 FPGA (LFE3-150EA-6FN1156C), DDR memories, a Lattice XP2-8, a USB device controller, power management electronics and a USB device controller. The various elements and their interconnections are shown in Figure I.7: LT-125 block diagram.

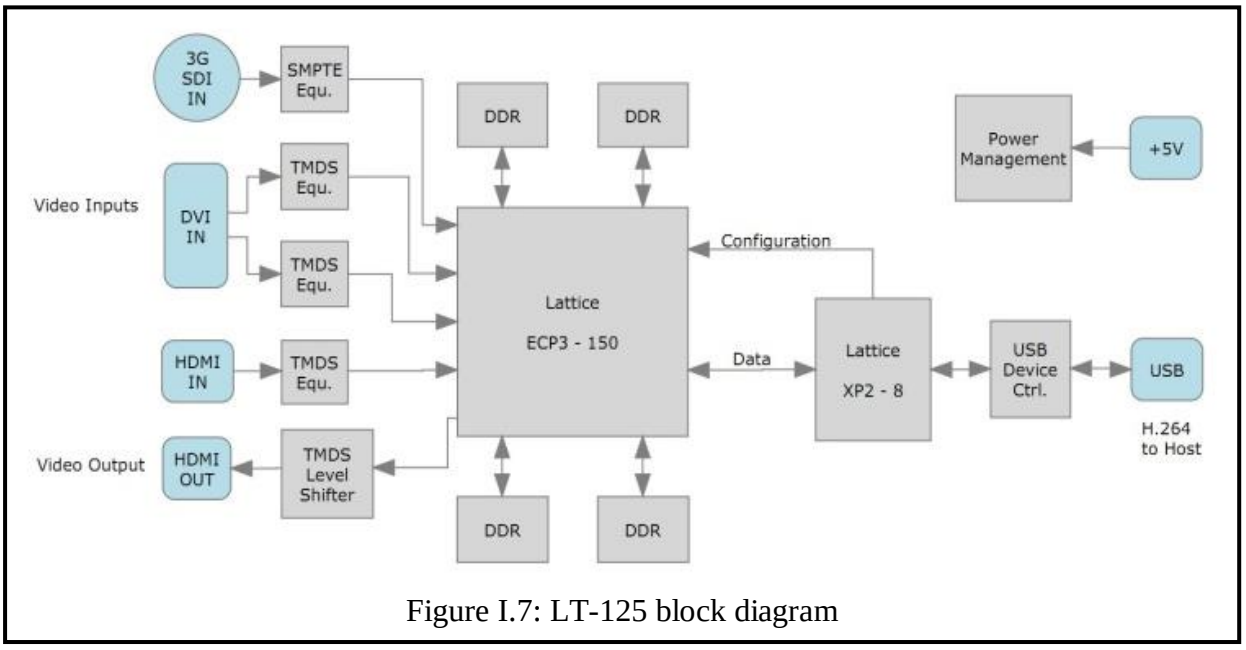


Figure I.7: LT-125 block diagram

When high speed video signal passes through a cable, signal degradation occurs. The LT-125 is equipped with input equalizers to restore video signal quality. ST Microelectronics TMDS equalizers ICs are used for DVI and HDMI inputs. These devices are designed to handle video at rates exceeding 1080p60 and also provide the necessary signal level shifting required by the FPGA. A Gennum 3G-SDI input equalizer is used to assure SMPTE signal integrity. The HDMI video output uses a ST Microelectronics TMDS level shifter as the FPGA output does not provide the required HDMI levels directly.

The Lattice ECP3-150 FPGA has access to four Low Power Mobile DDRs. These are 32-bit DDRs with a capacity of 256 Mbps each that operate at up to 133 MHz using Enciris Technologies' proprietary video optimized DDR controller IP.

A Lattice XP2 FPGA is used to transfer packets of data (e.g. compressed, uncompressed video, and parameters) to and from the USB device controller, and to provide the configuration bitstream of Lattice EP3-150 FPGA. The configuration bitstream is uploaded from the host PC via USB each time the board is used. This method allows for quick changes of configuration without requiring the use of the JTAG port and a permanent ECP3 configuration device (i.e. Flash memory). Typically, the Lattice ECP3-150 FPGA takes only a second or two to be reconfigured.

I.3.2 LT-125 board's features and applications

The most significant features of LT-125 board are:

- A platform for evaluating Enciris Technologies H.264 and VC-1 video compression and decompression IPs.
- Uses a powerful Lattice ECP3 FPGA with 150 KLUTs for video processing.

- Equipped with an SMPTE 3G/HD/SD/SDI video input, a DVI single and dual link video input, an HDMI video input, and an HDMI video output.
- FPGA configuration bitstream is uploaded directly via USB. JTAG is not required.
- Includes a PC-based application for Windows XP/Vista/7 32/64bit and Linux and Mac OSX for quick evaluation and FPGA configuration bitstream.
- A simple SDK that includes DirectShow is available for developing custom applications around the LT-125.

LT125 is used for many applications such as:

- HDTV capture and storage
- Video medical systems
- Video surveillance systems
- Internet broadcasting and teleconferencing.

I.3.3 DVI IN and HDMI OUT bypass demo of LT-125 board

In this section, the DVI IN and HDMI OUT bypass configuration demonstration on the LT-125 board is presented. Enciris uses three different software in the development flow of an FPGA: Altium Designer, Synplify Pro and Lattice Diamond. Moreover, a proprietary Python executable is used to generate and configuration the bitstream file. The flow of bitstream implementation on LT-125 Enciris board are shown in Figure I.8.

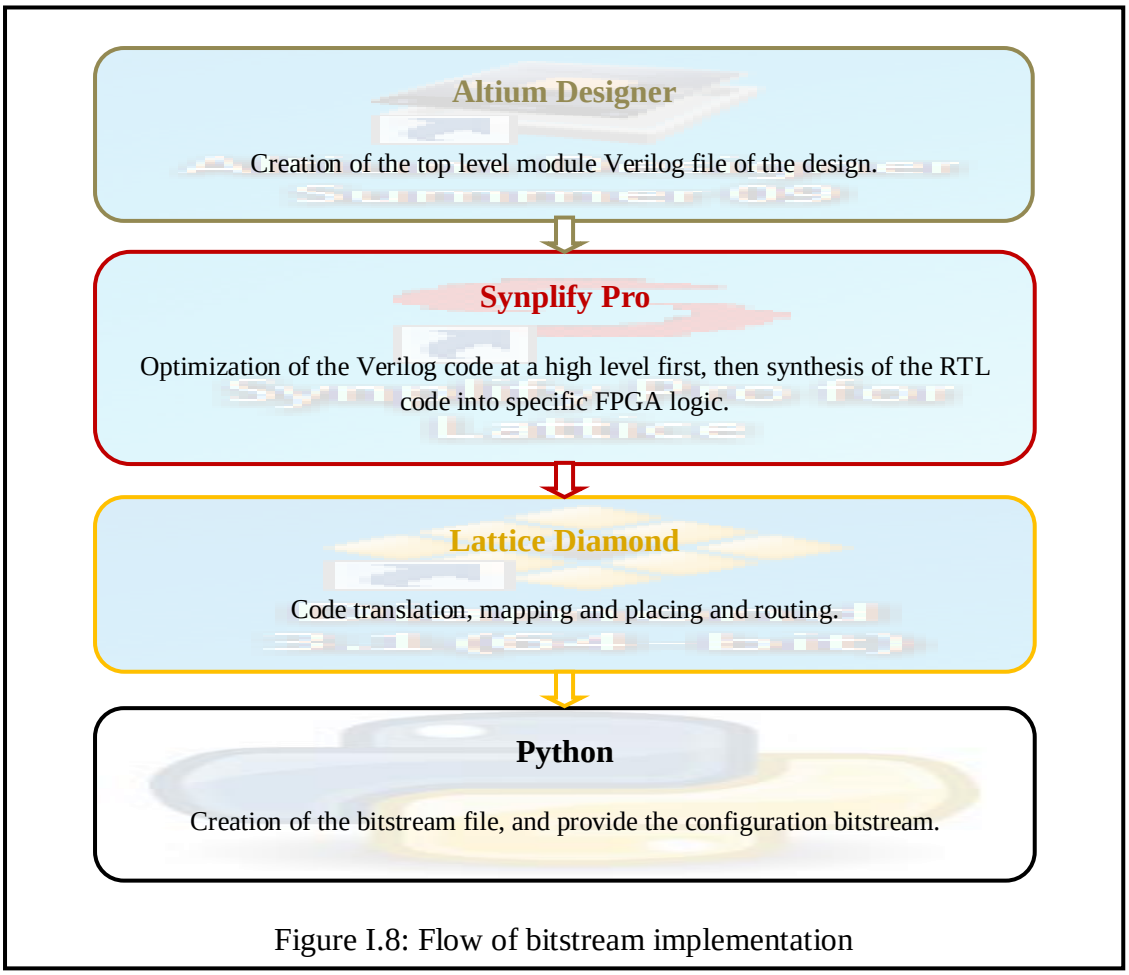


Figure I.8: Flow of bitstream implementation

The DVI IN and HDMI OUT bypass demo is shown in Figure I.9.

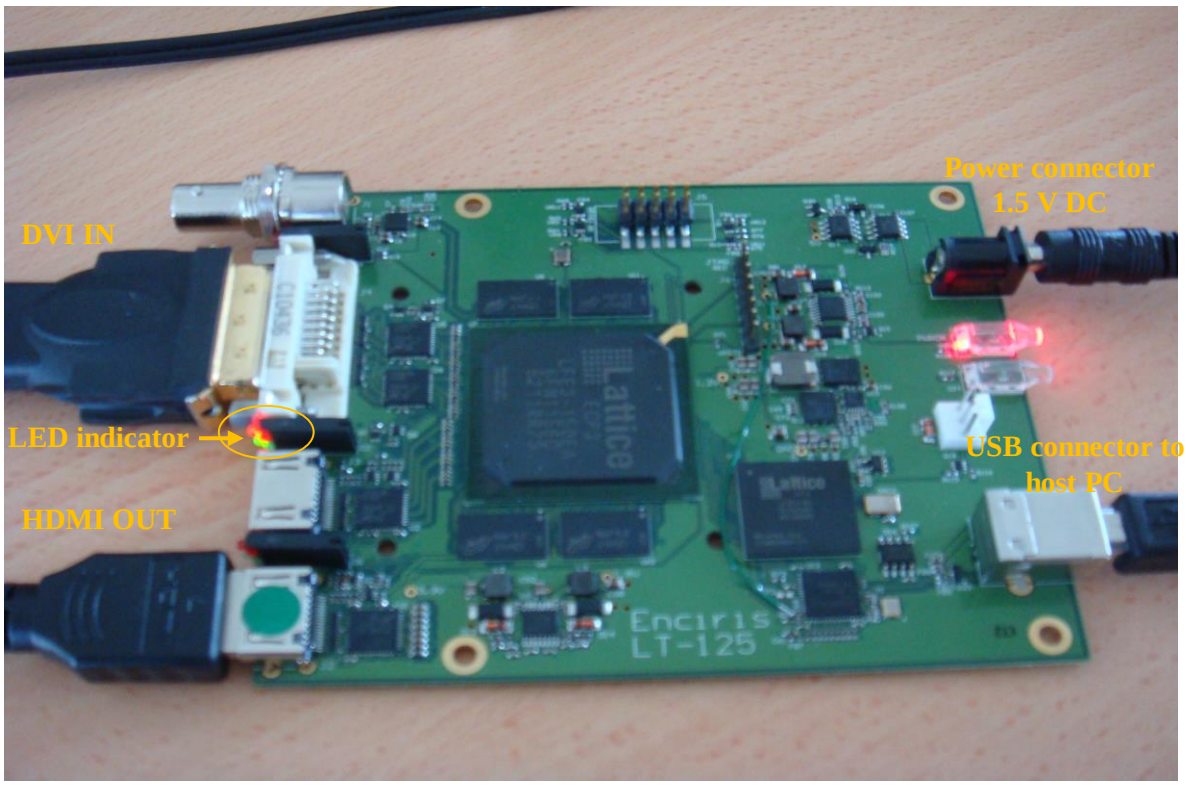


Figure I.9: DVI IN and HDMI OUT bypass demo

I.4 Implementation of the tri-rate SDI PHY IP core on the Enciris LT-125 board

In this section, the implementation of the SMPTE 3G/HD/SD/SDI video input functionality on the Enciris LT-125 board by using the Lattice tri-rate SDI PHY IP core will be described. The SDI scheme on the LT-125 board is shown in Figure I.10.

The SERDES block performs de-serialization outputting the parallel data to the tri-rate SDI IP core. The IP core performs the rate detection in order to determine the SDI configuration of the incoming video stream. Rate detection is performed by sequentially scanning the input for different SDI standards: 270 Mbps SD-SDI, 1.485 Gbps HD-SDI and 2.97 Gbps 3G-SDI interfaces. If the incoming video stream matches one of the SDI standards, the SDI IP core receiver locks to this video stream by asserting the *vid_active* signal, and outputs the 20-bit SD parallel data with format 4:2:2 and other control signals such as field, vblank, hblank, frame format and video format.

The *Sync-signals* block, *SDI 422 to 444* block and *YCrCb to RGB converter* block are used to convert SD parallel data to HDMI format.

The Cross-bar switch module multiplexes the video inputs to the various processing units. Note that at most two out of the three video inputs can be encoded in parallel while one input is bypassed.

The HDMI Bypass module forwards the incoming video without any further processing to the HDMI output.

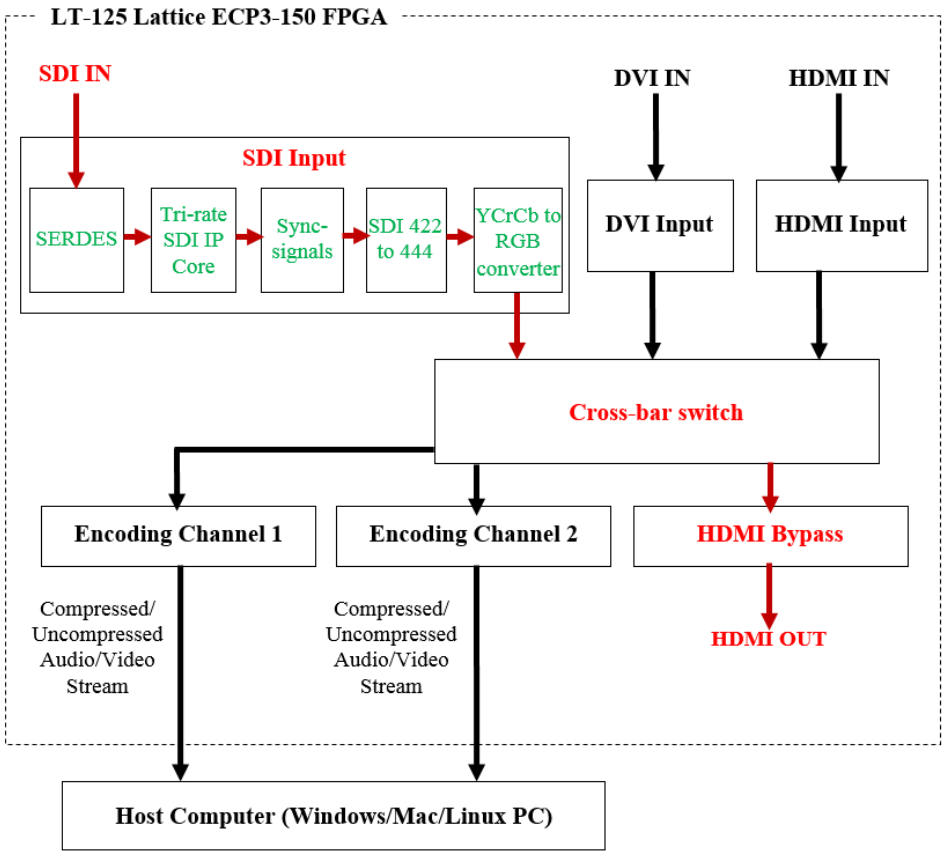


Figure I.10: SDI scheme on the LT-125 board

The tri-rate SDI PHY IP core demonstration setting on the Enciris LT-125 board design is shown in Figure I.11.

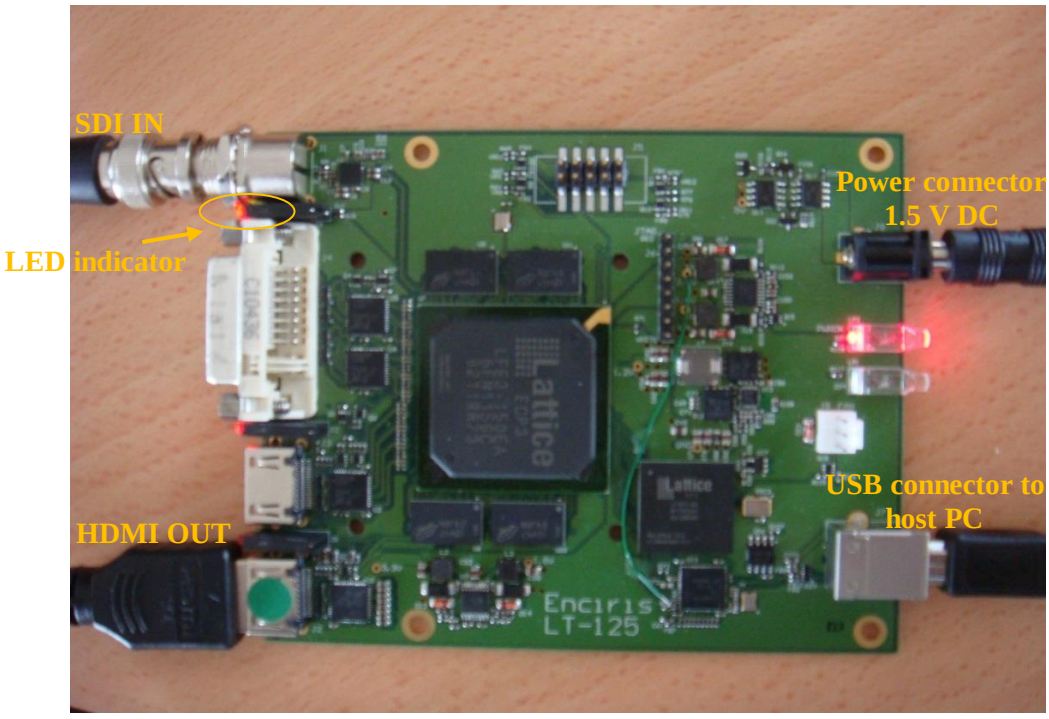


Figure I.11: Tri-rate SDI PHY IP core on LT-125 Enciris board design

I.5 Conclusion

At the end of this chapter, we have achieved our goal on the implementation of this IP core on the Enciris LT-125 board, provided an SMPTE 3G/HD/SD SDI video input functionality on the board, by understanding the LatticeECP3 SERDES/PCS, the Lattice tri-rate SDI PHY IP core, the LT-125 Enciris board, and particularly the Tri-Rate SDI PHY IP pass-through sample design on lattice board and DVI IN and HDMI OUT bypass demo on LT-125 board.

PART II

CHAPTER II
“H.264/AVC BITSTREAM DECODER”

CHAPTER II

"H.264/AVC BITSTREAM DECODER"

H.264/AVC is the newest and the most popular video compression/decompression standard of the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group. Compared to previous coding standards, it is able to deliver higher video quality for a given compression ratio, and better compression ratio for the same video quality. In other words, it allows reducing the required storage space while maintaining video quality.

This chapter will present:

- overview of video processing
- overview of H.264/AVC decoder
- H.264/AVC parameters - profile and level
- H.264/AVC bitstream decoder

II.1 Overview of video processing

This section presents the sufficient background information underlying the context and significance of digital video processing algorithms.

Digital image is a 2-D signal which is composed of very small picture elements called pixels.

Pixel is a small rectangular area which has a uniform intensity value. A higher visual quality is achieved with more pixels on the image. Pixel information can be coded using the fundamental colors in *RGB* (red, green and blue) color space or using luminance and chrominance in the *YCbCr* color space: *luminance* (*Y*), *chrominance blue* (*Cb*) and *chrominance red* (*Cr*). The human eye is less sensitive to color information than luminous information, thus video systems usually represent the pixel information using *YCbCr* color space because it facilitates the subsampling of color information when compared to *RGB*. Pixels represented in the *YCbCr* color space can be subsampled in order to reduce the information needed to store an image by up to 50%. High quality pictures are represented in 4:4:4 format, while 4:2:0 is used in video systems to compress images by a factor of 2 with a 4:1 subsampling on each chrominance component.

Video is a sequence of images. Each image is called *frame* and each frame has equal displaying time. Increasing capture and exhibition rates of video images (frames per second) leads a sensation of real motion.

Each frame is decomposed into two fields:

- Even Field: includes the even rows of the frame
- Odd Field: includes the odd rows of the frame

There are two types of scanned fields:

- *Progressive* scanned field: the even and odd fields are captured at the same time
- *Interlaced* scanned field: the even and the odd fields are captured at different times

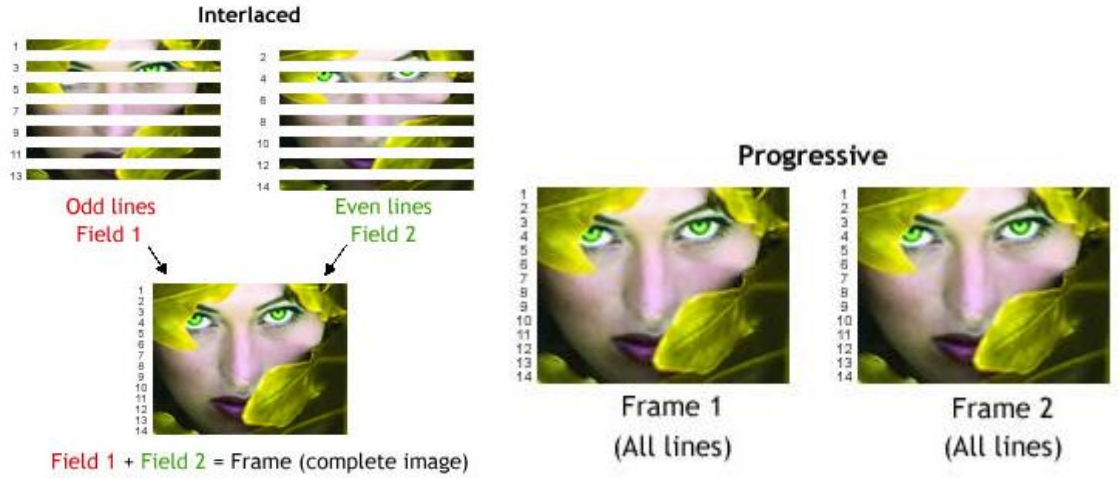


Figure II.1: Interlaced vs progressive scan

Each Frame is segmented into blocks of pixels. Most common regions of pixels used in image processing are: 4×4, 8×8, 8×16, 16×8 and 16×16 pixels. The blocks of pixels is called a *macroblock* of pixel samples, or simply *macroblock*. A group of macroblocks is called a *slice*. For example, a 16x16 Macroblock (MB) represented in *YCbCr* 4:2:0 consists of:

- 256 luminance *Y* pixels or one 16×16 macroblock in *Y* component which is composed of sixteen 4x4 sub-macroblocks in *Y* component
- 64 chroma *Cb* pixels or one 8×8 macroblock in *Cb* component which is composed of four 4x4 sub-macroblocks in *Cb* component
- 64 chroma *Cr* pixels or one 8×8 macroblock in *Cr* component which is composed of four 4x4 sub-macroblocks in *Cr* component

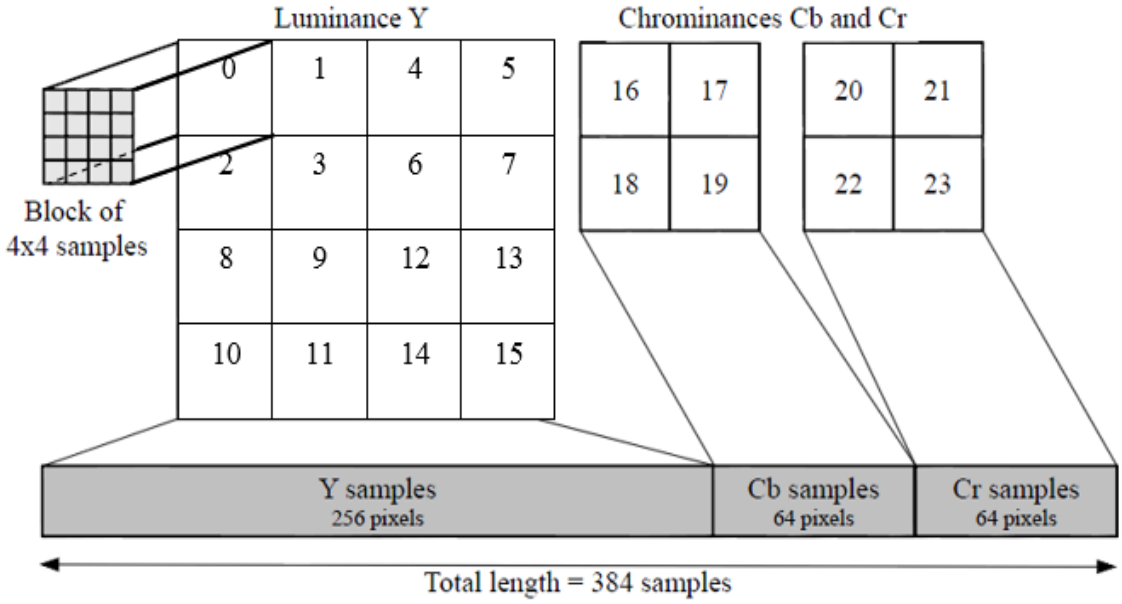


Figure II.2: Pixel array organization in a macroblock of 16x16 pixels

II.2 Overview of H.264/AVC decoder

The amount of raw video information to display on a high definition television screen is too big to be transmitted or stored at a reasonable cost. For that reason, we need a codec (encoder/decoder) pair to allow the video compression, maintaining image quality and reducing the amount of data. The encoder converts the source information in a compressed form before being transmitted or stored, and the decoder is responsible to convert the compressed information in video information again. The decoder has mechanisms to reconstruct the video content based on parameters sent by the encoder.

H.264/AVC encoding algorithm, which is the newest and the most popular video compression/decompression algorithm, was conceived to explore redundancies between successive frames and between blocks within a frame, using inter and intra frame prediction, a *DCT*-based transform, a quantization, deblocking filter and an entropy encoder mechanism to compress video data. [7] The decoding process in H.264/AVC which is the inverse process of encoder is shown in the Figure II.3.

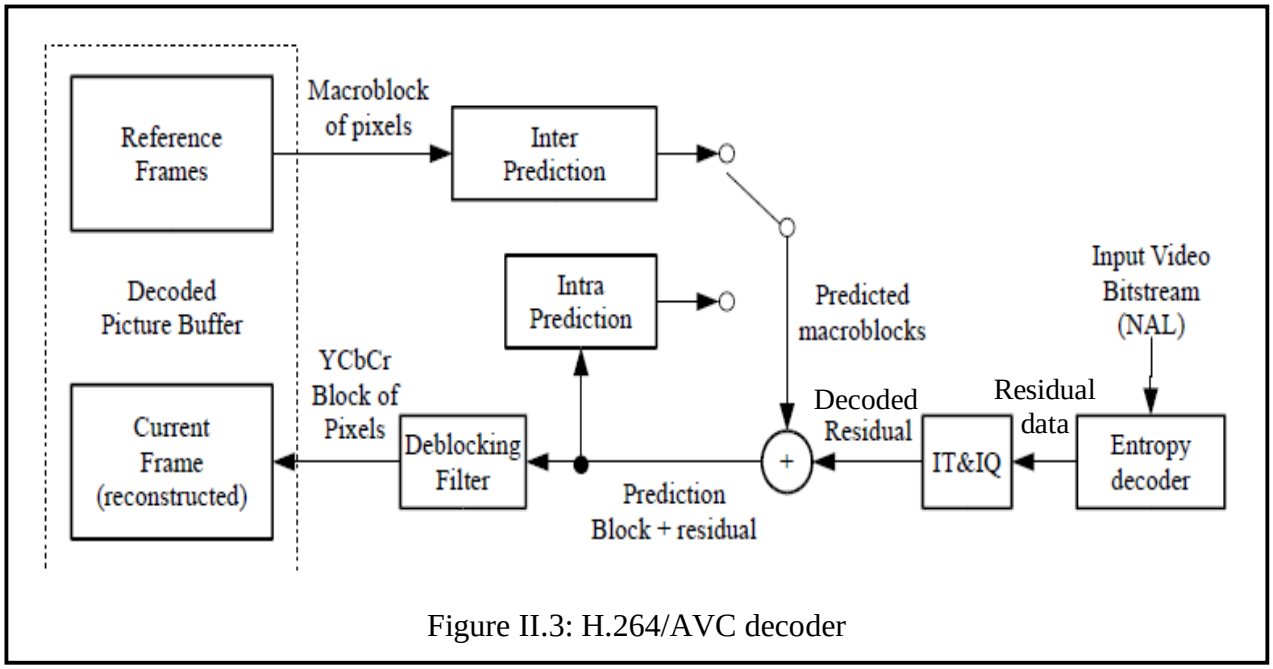


Figure II.3: H.264/AVC decoder

The compressed video bitstream is received in the video decoder within the *Network Abstraction Layer (NAL) unit*.

The *entropy decoder* or *bitstream decoder* contains a parser that receives a compressed video bitstream in the input from the *NAL* and decodes the quantized coefficients to generate the *residual data*. Also, it extracts the syntax elements to inter-frame and intra-frame prediction processes. The residual data is decoded using fixed or variable length binary codes in one of the entropy decoders: *Exp-Golomb, CAVLD* or *CABAD* decoder.

The residual data is then processed in the *Inverse Transform* and *Inverse Quantization* (IT and IQ) steps. Using information decoded from the bitstream, the decoder creates a *prediction block* or *decoded residual block* after the IT and IQ processing.

The H.264 Standard adopts two modes of block prediction: intra and inter prediction. *Inter prediction* refers to the reuse of information previously decoded frames stored in the decoded

picture buffer to predict current frame. *Intra prediction* reconstructs each image block from its previous coded block.

Finally, the *decoded residual block* is added to the predicted blocks of pixels, generating the pixel output that is filtered by the *deblocking filter*, smoothing block edges and improving the appearance of displayed images before exhibition.

II.3 H.264/AVC parameters - profile and level

The H.264/AVC encoder/decoder's capabilities is specified by a *profile* and *level*. A *profile* defines a set of coding tools or algorithms that can be used in generating a conforming bitstream, whereas a level places constraints on certain key parameters of the bitstream. In other words, a profile defines specific encoding techniques that you can or can't utilize when encoding the files (such as *B-frames*), while the *level* defines details such as the maximum resolutions and data rates. The profiles is categorized in to 4 main classes: *constrained baseline profile*, *baseline profile*, *main profile* and *extended profile* as shown in Figure II.4. The higher profiles provides more functionalities hence the better quality but increasing both decoding complexity and encoding time.

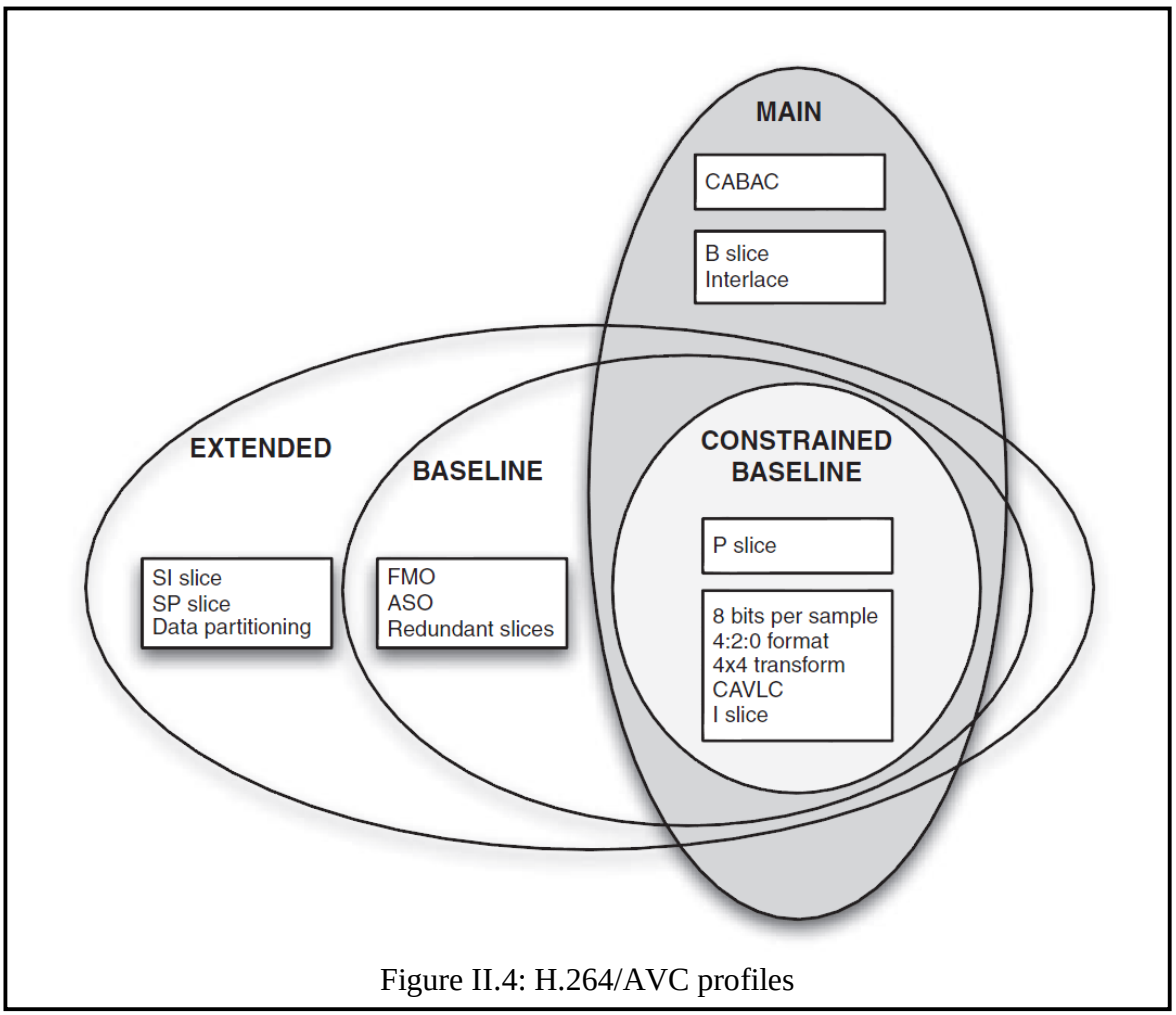


Figure II.4: H.264/AVC profiles

Table II.1: H.264/AVC levels

Level number	Max macroblock processing rate MaxMBPS (MB/s)	Max frame size MaxFS (MBs)	Max decoded picture buffer size MaxDpbMbs (MBs)	Max video bit rate MaxBR (1000 bits/s, 1200 bits/s, cpbBrVclFactor bits/s, or cpbBrNalFactor bits/s)	Max CPB size MaxCPB (1000 bits, 1200 bits, cpbBrVclFactor bits, or cpbBrNalFactor bits)	Vertical MV component range MaxVmvR (luma frame samples)	Min compression ratio MinCR	Max number of motion vectors per two consecutive MBs MaxMvsPer2Mb
1	1 485	99	396	64	175	[−64,+63.75]	2	-
1b	1 485	99	396	128	350	[−64,+63.75]	2	-
1.1	3 000	396	900	192	500	[−128,+127.75]	2	-
1.2	6 000	396	2 376	384	1 000	[−128,+127.75]	2	-
1.3	11 880	396	2 376	768	2 000	[−128,+127.75]	2	-
2	11 880	396	2 376	2 000	2 000	[−128,+127.75]	2	-
2.1	19 800	792	4 752	4 000	4 000	[−256,+255.75]	2	-
2.2	20 250	1 620	8 100	4 000	4 000	[−256,+255.75]	2	-
3	40 500	1 620	8 100	10 000	10 000	[−256,+255.75]	2	32
3.1	108 000	3 600	18 000	14 000	14 000	[−512,+511.75]	4	16
3.2	216 000	5 120	20 480	20 000	20 000	[−512,+511.75]	4	16
4	245 760	8 192	32 768	20 000	25 000	[−512,+511.75]	4	16
4.1	245 760	8 192	32 768	50 000	62 500	[−512,+511.75]	2	16
4.2	522 240	8 704	34 816	50 000	62 500	[−512,+511.75]	2	16
5	589 824	22 080	110 400	135 000	135 000	[−512,+511.75]	2	16
5.1	983 040	36 864	184 320	240 000	240 000	[−512,+511.75]	2	16

II.4 H.264/AVC bitstream decoder

II.4.1 H.264/AVC bitstream format

Obviously, the decoder operates with a sequence of bits received in a specific format. The byte stream format puts a synchronization byte sequence (0×00000001) before every *NAL* (*Network Abstraction Layer*) unit packet, as shown in Figure II.5.

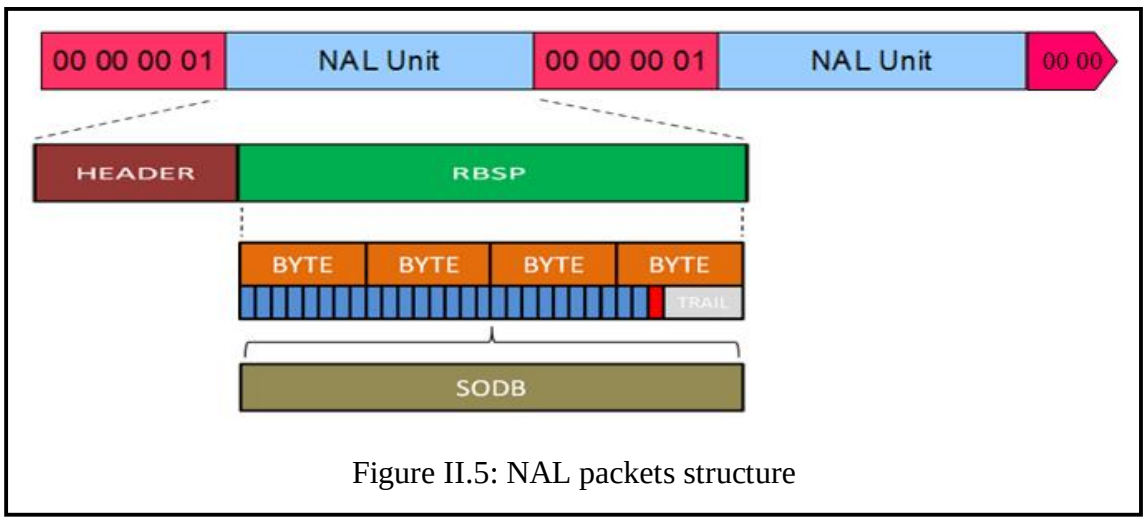


Figure II.5: NAL packets structure

The first byte of a *NAL* unit packet is a header that contains information about the type of *NAL* packet which is shown in Table II.2. As can be seen from the Figure II.5, the payload of *NAL* packet identified as *RBSP* (*Raw Byte Sequence Payload*). *RBSP* describes a row of bits specified order of *SODB* (*String Of Data Bits*).

Table II.2: *NAL* types

Type (5 bits LSB of the first byte in a *NAL* packet)	**Definition**
0	Undefined
1	Non-IDR slice
2	Slice data partition A layer
3	Slice data partition B layer
4	Slice data partition C layer
5	IDR slice
6	Additional information (SEI)
7	Sequence parameter set
8	Picture parameter set
9	Access unit delimiter
10	End of sequence
11	End of stream
12	Filler data
13..23	Reserved
24..31	Undefined

The detailed H.264 data stream is shown in Figure II.6.

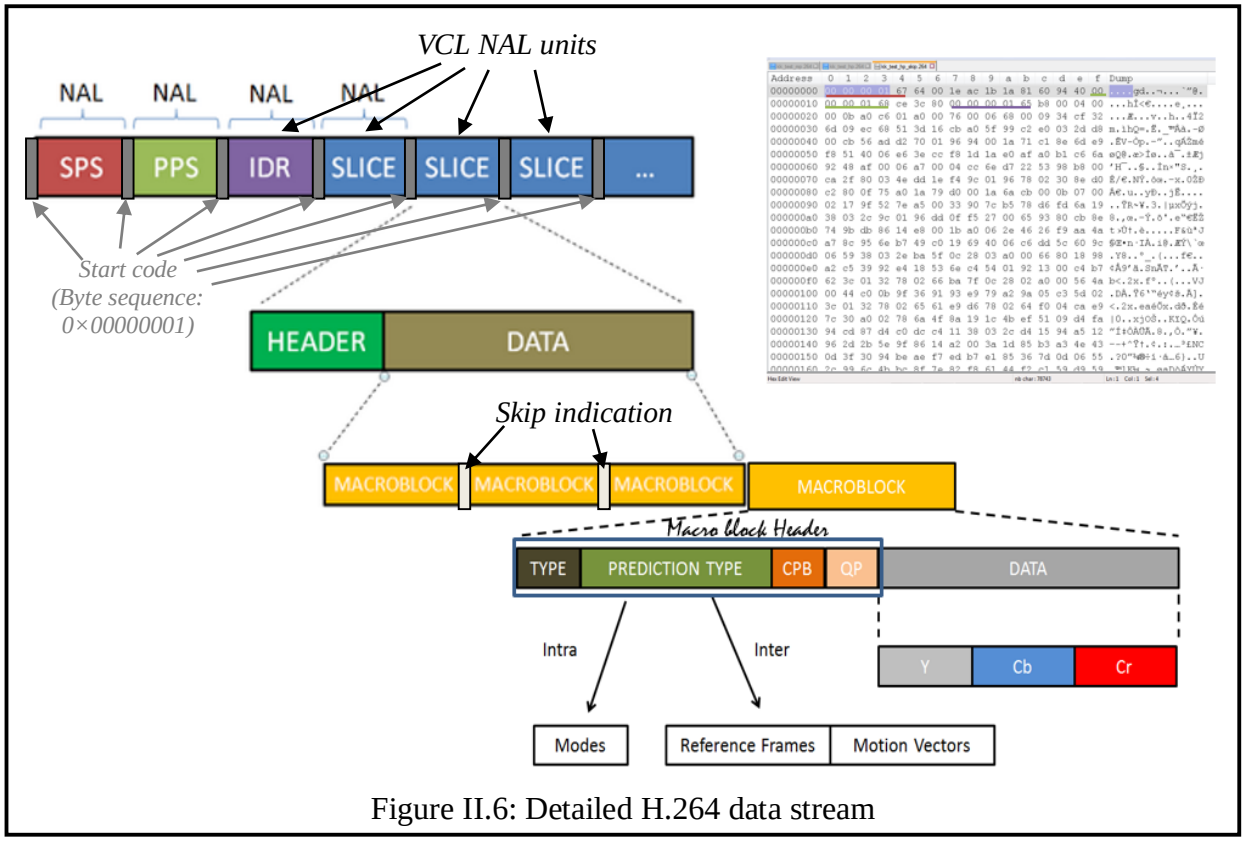


Figure II.6: Detailed H.264 data stream

The *Network Abstraction Layer* (*NAL*) consists of a series of *NAL* units: *SPS*, *PPS*, *IDR*, and *SLICEs*. *Sequence Parameter Sets* (*SPS*) and *Picture Parameter Sets* (*PPS*) are *NAL* units that signal certain common control parameters to the decoder. For instance, a *Sequence Parameter Set* (*SPS*) contains parameters that are applied to a complete video sequence such as: the picture order count, decoded picture width and height and the choice of progressive or interlaced (frame or frame/field) coding. A *Picture Parameter Set* (*PPS*) contains parameters that are applied to the current decoded picture such as: an picture identifier, a flag to select *CAVLD* or *CABAD* entropy decoding; the number of reference pictures in list 0 and list 1 that may be used for prediction, initial quantized parameters among others.

A coded video sequence begins with an *Instantaneous Decoder Refresh* (*IDR*) access unit, made up of one or more IDR slices, a special type of Intra coded slice. Subsequent video frames or fields, described as Access Units, are coded as slices. The video sequence ends when a new *IDR* slice is received, signaling a new coded sequence, or when the transmission is complete.

Coded video data is communicated in *Video Coding Layer* (*VCL*) *NAL* units, known as coded slices. An access unit, a coded frame or field, is made up of one or more slices. At the slice layer, each slice consists of a Slice Header and Slice Data. The Slice Data is a series of coded macroblocks (*MB*) and *skip macroblock indicators* which signal that certain macroblock positions contain no data. Each coded macroblock contains the following syntax elements [5]:

- Macroblock header
  - MB type: *I/intra* coded, *P/inter* coded from one reference frame, *B/inter* coded from one or two reference frames.
  - Prediction information: prediction mode(s) for an *I* macroblock, choice of reference frame(s) and motion vectors for a P or B macroblock.
  - Coded Block Pattern (*CBP*): indicates which luma (*Y*) and chroma (*Cb*, *Cr*) blocks contain non-zero residual coefficients.
  - Quantization Parameter (*QP*), for macroblocks with $CBP \neq 0$.
- Macroblock data or Residual data, for blocks containing non-zero residual coefficients.

II.4.2 Coding

A coded H.264 stream or an H.264 file consists of a series of coded symbols. In H.264/AVC standard, the entropy decoder or bitstream decoder contains several methods for decoding the coded symbols to generate the residual data. These methods are as follows [5]:

- *Fixed length decoding*: a symbol is extracted from a binary code with a specified fixed length (*n* bits).
- *Exponential-Golomb variable length decoding*: a symbol is extracted from a binary code with a varying number of bits (*v* bits). In general, at encoder side, shorter *Exp-Golomb* codewords are assigned to symbols that occur more frequently.
- *CAVLD*: Context-Adaptive Variable Length Decoding, a specially-designed method of decoding transform coefficients in which different sets of variable-length codes are extracted depending on the statistics of recently-coded coefficients, using context adaptation.
- *CABAD*: Context-Adaptive Binary Arithmetic Decoding, a method of arithmetic decoding in which the probability models are updated based on previous coding statistics.

Syntax elements/data occurring in the bitstream/syntax above the slice data level are decoded using *Fixed Length decoding* or *Exp-Golomb decoding*. Syntax elements/data at the slice data level and below are decoded using *CAVLD* or *CABAD decoding*. The *Exp-Golomb decoding* and *CAVLD* will be described further in the following sections.

II.4.2.1 Exp-Golomb decoding

Exp-Golomb decoding uses smaller codeword length for frequently occurring data and larger codeword length for less frequently occurrences. As a result, the average codeword length is reduced and higher compression is achieved. An Exp-Golomb codeword has the following structure: [M zero] [1] [INFO], where M denotes the number of leading zero and INFO denotes an M-bit field of information. Table II.3 and Table II.4 list the first few signed Exp-Golomb codewords and unsigned Exp-Golomb codewords respectively.

Table II.3: Signed Exp-Golomb codewords

code_num	Codeword
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
...	...

Table II.4: Unsigned Exp-Golomb codewords

code_num	Codeword
0	1
1	010
-1	011
2	00100
-2	00101
3	00110
-3	00111
4	0001000
-4	0001001
...	...

II.4.2.2 Context-Adaptive Variable Length Decoding, CAVLD

Context-Adaptive Variable Length Decoding (*CAVLD*) is the inverse process of *CAVLC*, or a type of run length decoding, where the number of zeros is increased or reconstructed by a run length parameter that is transmitted by *CAVLC*. The algorithm *CAVLC* provides better efficiency but increasing decoding complexity in compression. During video compression, many video coefficients become zero after the quantization step, which is termed a run of zeros. Instead of encoding each zero into the video compression stream, run length compression is used, where the run length of the zeros is encoded to increase the overall compression efficiency.

CAVLC is designed to take advantage of several characteristics of quantized 4x4 blocks [5]:

- After prediction, transformation and quantization, blocks are typically sparse (containing mostly zeros).
- The highest non-zero coefficients after zig-zag scan are often sequences of +/- 1. *CAVLC* signals the number of high-frequency +/-1 coefficients in a compact way.
- The number of non-zero coefficients in neighboring blocks is correlated. The number of coefficients is decoded using a look-up table; the choice of look-up table depends on the number of non-zero coefficients in neighboring blocks.
- The level (magnitude) of non-zero coefficients tends to be higher at the start of the reordered array (near the *DC* coefficient) and lower towards the higher frequencies. *CAVLC* takes advantage of this by adapting the choice of *VLC* look-up table for the "level" parameter depending on recently coded level magnitudes.

Figure II.7 shows the flowchart for *CAVLC* codec: (a) Decoder and (b) Encoder. Table II.5 describes *CAVLC*/*CAVLD* decoder syntax elements. Table II.6 indicates the choice of look-up table for *coeff_token*.

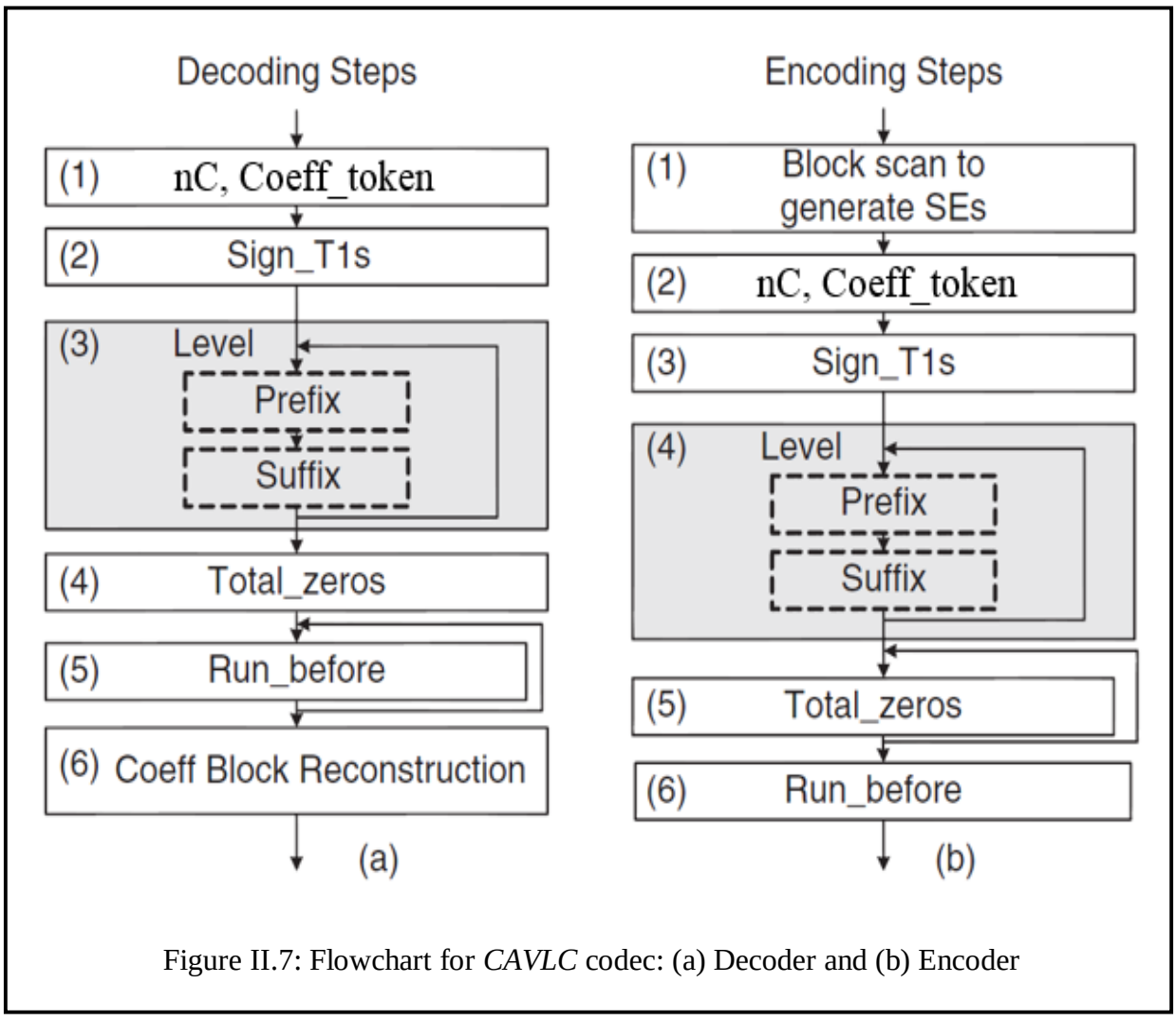


Figure II.7: Flowchart for *CAVLC* codec: (a) Decoder and (b) Encoder

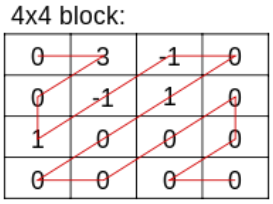
Table II.5: *CAVLC/CAVLD* decoder syntax elements

Syntax Elements	Description
nC	the parameter to choose the look-up table for *coeff_token*
coeff_token	the number of all non-zero coefficients (*TotalCoeff*) and the number of trailing ones (*T1s*) are encoded by this syntax element
trailing_ones_sign_flag	the sign bit of each *T1* is reverse zig-zag scan order is encoded by this syntax element
level	The value of each non-zero coefficient (except for *T1s*) is encoded by this syntax element
total zeros	The total number of zero coefficients preceding the last non-zero coefficients in zig-zag order is encoded by this syntax element
run before	The number of successive zero coefficients following the non-zero coefficients in reverse zig-zag order.

Table II.6: Choice of look-up table for *coeff_token*

nC	Table for coeff_token
0, 1	VLC table 1
2, 3	VLC table 2
4, 5, 6, 7	VLC table 3
8 or above	FLC

In the following examples, we assume that table *Num-VLC0* [6] is used to encode *coeff_token*.



0,3,0,1,-1,-1,0,1,0…
TotalCoeffs = 5, indexed from highest frequency (4) to lowest frequency (0)
TotalZeros = 3
T1s = 3 (in fact there are 4 trailing ones but only 3 can be encoded as a “special case”)

Encoding:

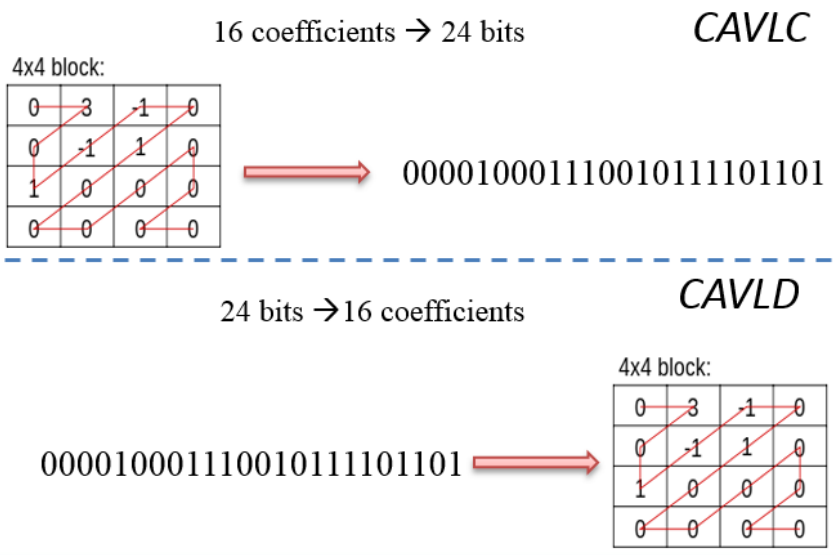
Element	Value	Code
coeff_token	*TotalCoeffs* = 5, *T1s* = 3	0000100
T1 sign (4)	+	0
T1 sign (3)	-	1
T1 sign (2)	-	1
Level (1)	+1 (use *Level_VLC0*)	1
Level (0)	+3 (use *Level_VLC1*)	0010
TotalZeros	3	111
run_before(4)	*ZerosLeft* = 3; *run_before* = 1	10
run_before(3)	*ZerosLeft* = 2; *run_before* = 0	1
run_before(2)	*ZerosLeft* = 2; *run_before* = 0	1
run_before(1)	*ZerosLeft* = 2; *run_before* = 1	01
run_before(0)	*ZerosLeft* = 1; *run_before* = 1	No code required; last coefficient.

The transmitted bitstream for this block is 000010001110010111101101.

Decoding: The output array is "built up" from the decoded values as shown below. Values added to the output array at each stage are underlined.

Code	Element	Value	Output array
0000100	*coeff_token*	*TotalCoeffs* = 5, *T1s* = 3	Empty
0	*T1* sign	+	1
1	*T1* sign	-	-1, 1
1	*T1* sign	-	-1, -1, 1
1	*Level*	+1	1, -1, -1, 1
0010	*Level*	+3	3, 1, -1, -1, 1
111	*TotalZeros*	3	3, 1, -1, -1, 1
10	*run_before*	1	3, 1, -1, -1, 0, 1
1	*run_before*	0	3, 1, -1, -1, 0, 1
1	*run_before*	0	3, 1, -1, -1, 0, 1
01	*run_before*	1	3, 0, 1, -1, -1, 0, 1

The decoder has inserted two zeros; however, *TotalZeros* is equal to 3 and so another 1 zero is inserted before the lowest coefficient, making the final output array: 0, 3, 0, 1, -1, -1, 0, 1. After reconstruction a 4x4 block, we get the sixteen coefficients: 0, 3, 0, 1, -1, -1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0.



CHAPTER III
"H.264/AVC BITSTREAM DECODER HARDWARE DESIGN IN VERILOG MODELS"

CHAPTER III

"H.264/AVC BITSTREAM DECODER HARDWARE DESIGN IN VERILOG MODELS"

The bitstream decoder or entropy decoder is a block that handles the compressed video bitstream within the video decoder. The bitstream decoder is required to process the input bitstream, identify syntactic elements and route the associated data to the appropriate decoder module, like the inter-frame or the intra-frame prediction blocks.

This chapter will describe about:

- bitstream decoder hardware architecture
- tools used in the FPGA development flow design
- hardware implementation and results

III.1 Bitstream decoder hardware architecture

The H.264 decoder which is going to be designed can support the constrained baseline profile, level at 4 (video format 1080 HD and resolution 1920x1088 pixels). The implemented hardware architecture for the bitstream decoder corresponding to this decoder is presented in Figure III.1.

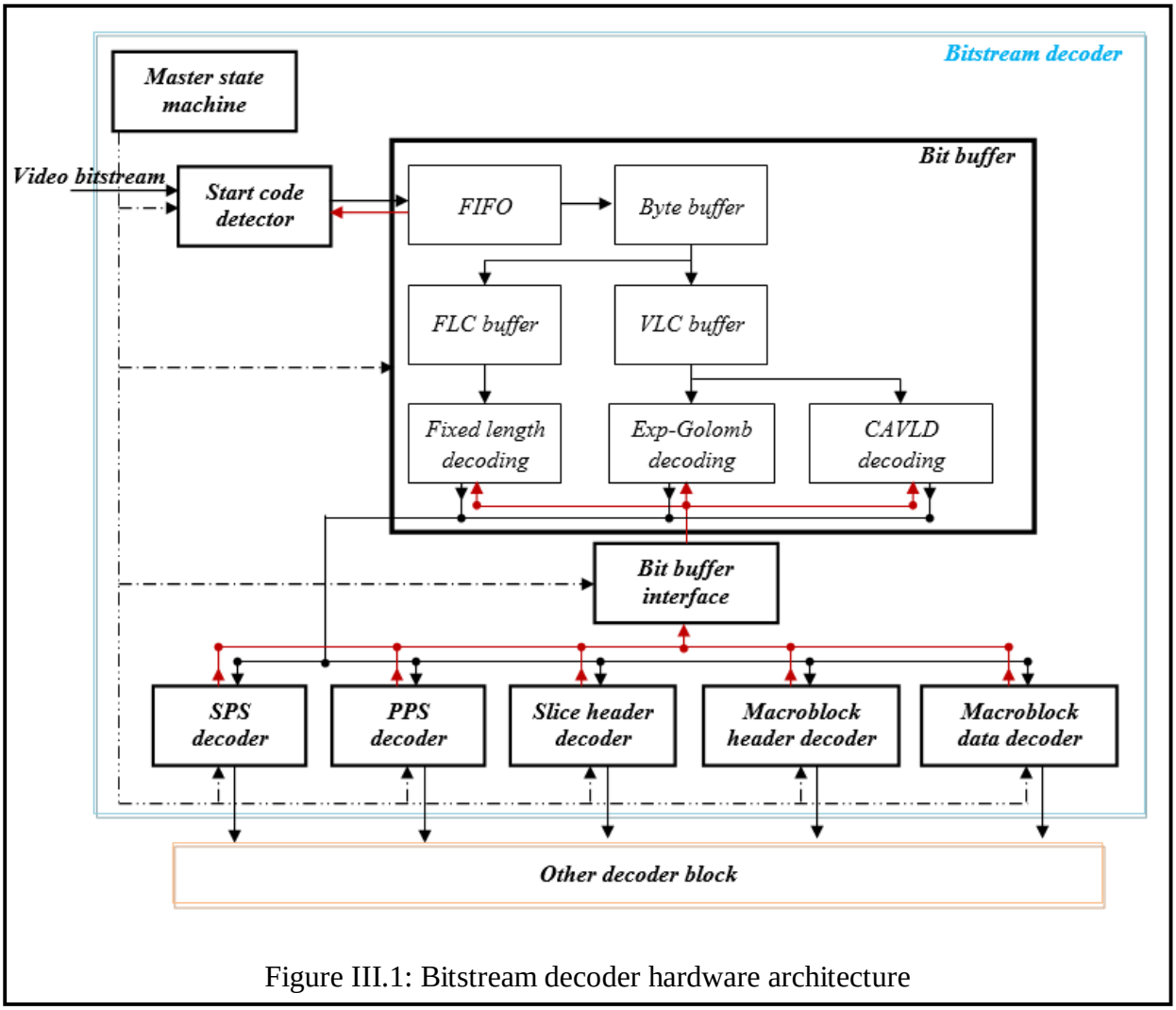


Figure III.1: Bitstream decoder hardware architecture

III.1.1 Flowchart of bitstream decoder

The bitstream decoder is the first processing step of decoder and it is a highly sequential process. As shown in Figure III.2, the compressed video bitstream is decoded as the following steps:

1. Detect the start code and decode *NAL* type: the byte sequence (0×00000001) and the first byte of the *NAL* packet
2.
   - if the *NAL* type is *SPS* (5 bits of the first byte in a *NAL* packet = 7) , decode *SPS* layer
   - if the *NAL* type is *PPS* (5 bits of the first byte in a *NAL* packet = 8), decode *PPS* layer
   - if the *NAL* type is *IDR slice or Non-IDR slice* (5 bits of the first byte in a *NAL* packet = 1 or 5), decode:
     a) Slice header
     b) Macroblock header
     c) Macroblock data
3. Send the decoded bitstream (residual data, and syntactic elements) to the appropriate decode module for further processing
4. Restart step 1 until the end of video bitstream.

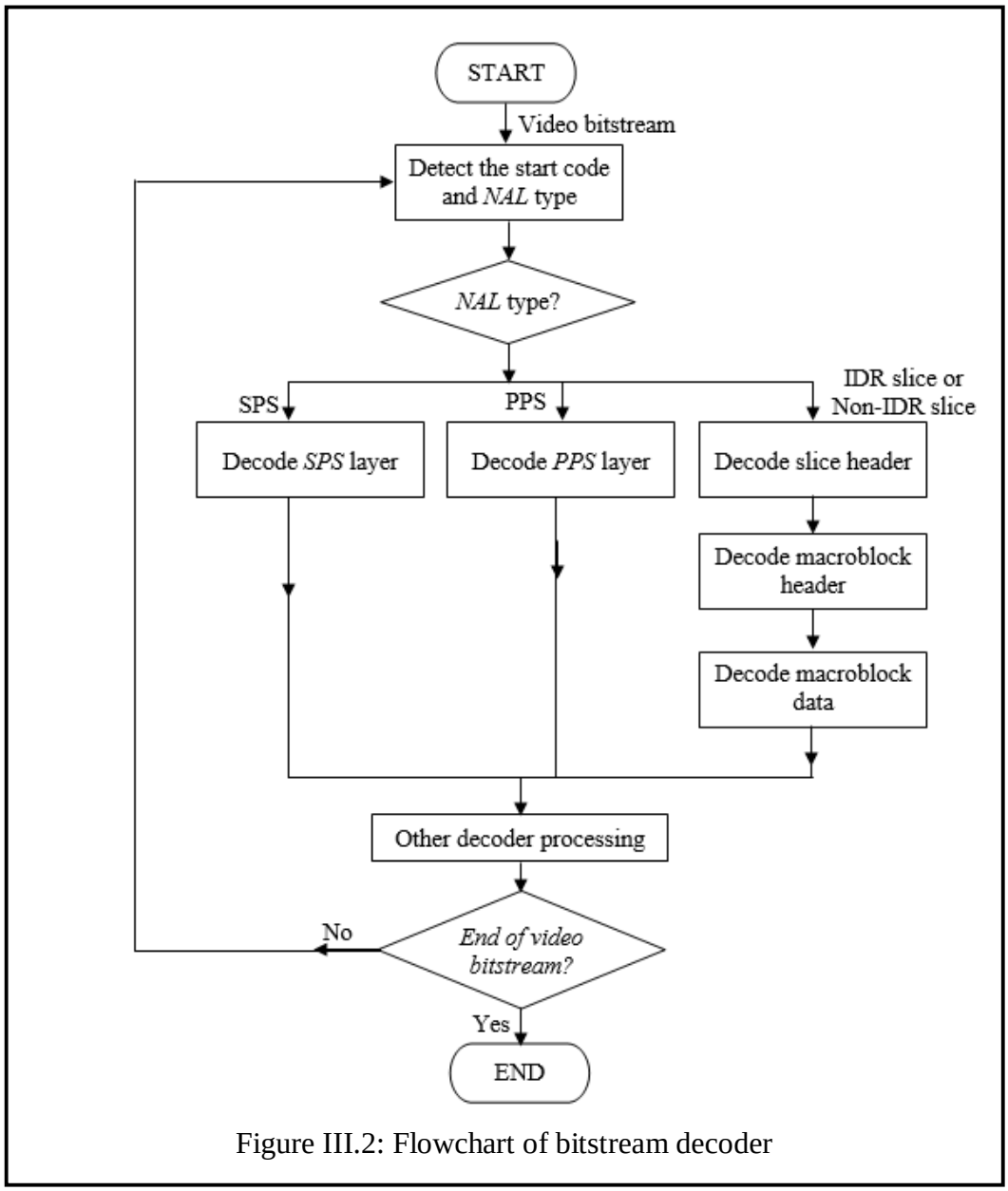


Figure III.2: Flowchart of bitstream decoder

III.1.2 Functionalities of each module

As shown in Figure III.1, the functionalities of each module are as follow:

- *Master state machine*: outputs the controlled signals to activate the modules for decoding the corresponded layer.

- *Start code detector*: reads byte by bytes from video bitstream, detects the start code (byte sequence 0×00000001), decodes *NAL* type (5 bits of the first byte in a *NAL* packet), and outputs the data one byte each cycle to the FIFO in bit buffer module starting from the first byte of *NAL* packet (excluded).

- *Bit buffer*: this module outputs the data which is decoded using whether fixed length decoding, *Exp-Golomb*, or *CAVLD* and it contains the following sub-modules:
  - *FIFO*: stores the output data from the *Start code detector* module
  - *Bytes buffer*: stores bytes from *FIFO*, aligns buffer position by shifting the bytes that has been decoded
  - *FLC buffer*: stores bytes from *Bytes buffer*, aligns buffer position by shifting the bits that has been decoded, is used for fixed length decoding
  - *VLC buffer*: stores bytes from *Bytes buffer*, aligns buffer position by shifting the bits that has been decoded, is used for Exp-Golomb decoding and CAVLD decoding
  - *Fixed length decoding*: decodes the syntax elements that is encode in fixed length code
  - *Exp-Golomb decoding*: decodes the syntax elements that is encode in Exp-Golomb code
  - *CAVLD* decoding: decodes the syntax elements that is encode in CAVLC code

- *Bit buffer interface*: is the multiplexer between the threes decoding: fixed length decoding, *Exp-Golomb*, or *CAVLD*.

- *SPS decoder*: decodes the syntax elements in sequence parameter set layer, outputs the decoded data such as: the picture order count, decoded picture width and height and the choice of progressive or interlaced (frame or frame/field) coding to the appropriate decoder module.

- *PPS decoder*: decodes the syntax elements in picture parameter set layer, outputs the decoded data such as: an picture identifier, a flag to select *CAVLD* or *CABAD* entropy decoding; the number of reference pictures in list 0 and list 1 that may be used for prediction, and initial quantized parameters to the appropriate decoder module.

- *Slice header decoder*: decodes the syntax elements in slice header layer, outputs the decoded data such as: slice type, frame number, picture order count type, and slice QP data to the appropriate decoder module (ex. macroblock header decoder module).

- *Macroblock header decoder*: decodes the syntax elements in macroblock header layer, outputs the decoded data such as: macroblock type, code block pattern, and prediction mode to the appropriate decoder module (ex. macroblock data decoder module).

- *Macroblock data decoder*: decodes the data elements or residual data in macroblock data layer, outputs the residue coefficients to the appropriate decoder module (ex. IT/IQ module).

Syntax elements/data occurring in the bitstream/syntax at *SPS*, *PPS*, *Slice header* and *Macroblock header* layer are decoded using *Fixed Length decoding* or *Exp-Golomb decoding*. Syntax elements/data at *Macroblock data* layer are decoded using *CAVLD decoding*.

III.2 References models and tools used in the FPGA development flow design

The references models that are used for helping understanding and coding H.264 hardware decoder are:

1. *Joint Model* (JM) ITU-T H.264 encoder and decoder model in C code as *Microsoft visual studio* project [8]
2. *Enciris H.264* encoder model in C code as *Microsoft visual studio* project
3. *Enciris VC1* decoder hardware model as Altium project
4. *Nova* Verilog H.264 decoder model [9]

The tools that are used in the FPGA development flow design are:

1. *Notepad++* : to create and edit the source codes
2. *Altium designer*: to compile the source codes and to create top level Verilog files
3. *ActiveHDL*: to do the simulation of Verilog files and verify the results
4. *Python*: to create *CAVLD ROM* table file in *.mem* for IP Express diamond which generates the *CAVLD ROM* table in Verilog file
5. *Microsoft visual studio, Joint Model* (JM) ITU-T H.264 encoder and decoder model in C code project: to create the trace file *.txt* of decoder
6. *Beyond Compare*: to compare and verify the results between the trace file created by the simulation in ActiveHDL and the trace file created by JM ITU-T C code model project
7. *Diamond 3.1*: to do the synthesize design, map design, place and route design

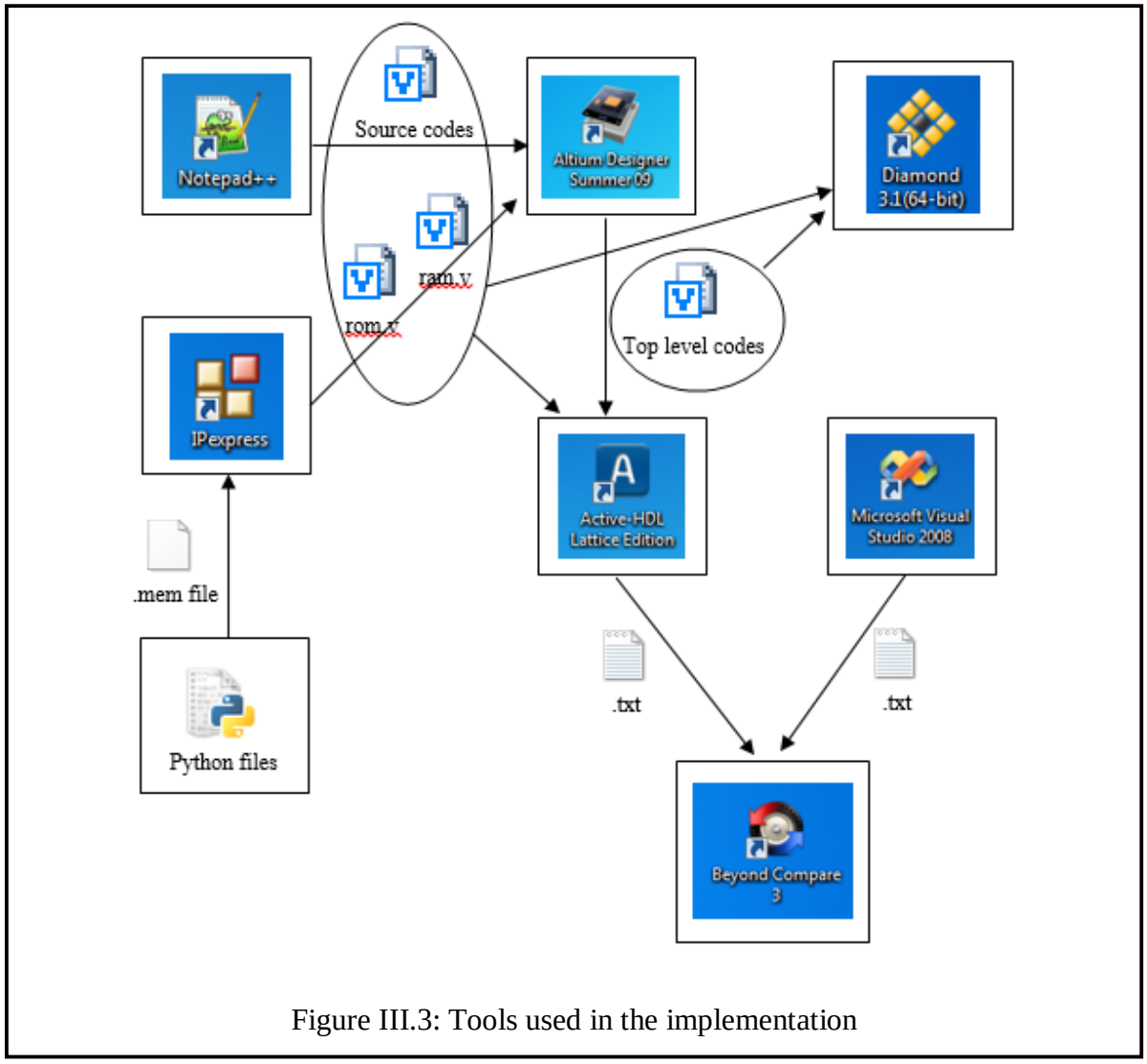


Figure III.3: Tools used in the implementation

III.3 Hardware Implementation

For the time being, the bitstream decoder is not completely implemented yet. Because of lacking time and because of the complexity of *CAVLD* decoding, the *CAVLD decoding* is not completely coded yet so that the bitstream decoder at the present can decode only the *SPS*, *PPS*, *Slice header*, and *Macroblock header* layer. And, each layer was decoded by the corresponding module which is coded as a state machine because the video stream input is a serial data and some syntax elements and data elements is encoded in variable length codes, *Exp-Golomb code and CAVLD code*. The following sections demonstrates how to implement the *Exp-Golomb decoding* and *CAVLD decoding*.

III.3.1 Implementation of Exp-Golomb decoding

As shown in Figure III.4, the code_num is decoded as follows:

1. Read a series of consecutive zeros until a 1 is detected. Count the number of zeros (M).
2. Read {[1], [M bits INFO]}.
3.
   - For unsigned Exp-Golomb codewords:
     code_num = {[1], [M bits INFO]} - 1
   - For signed Exp-Golomb codewords:
     - Read the Least Significant Bit (LSB) of {[1], [M bits INFO]}
       - If the LSB of {[1], [M bits INFO]} is equal to zero (positive number), then:
         code_num = {[1], [M bits INFO]} / 2
       - If the LSB of {[1], [M bits INFO]} is equal to one (negative number), then:
         code_num = Two's Complement of ({[1], [M bits INFO]} / 2)

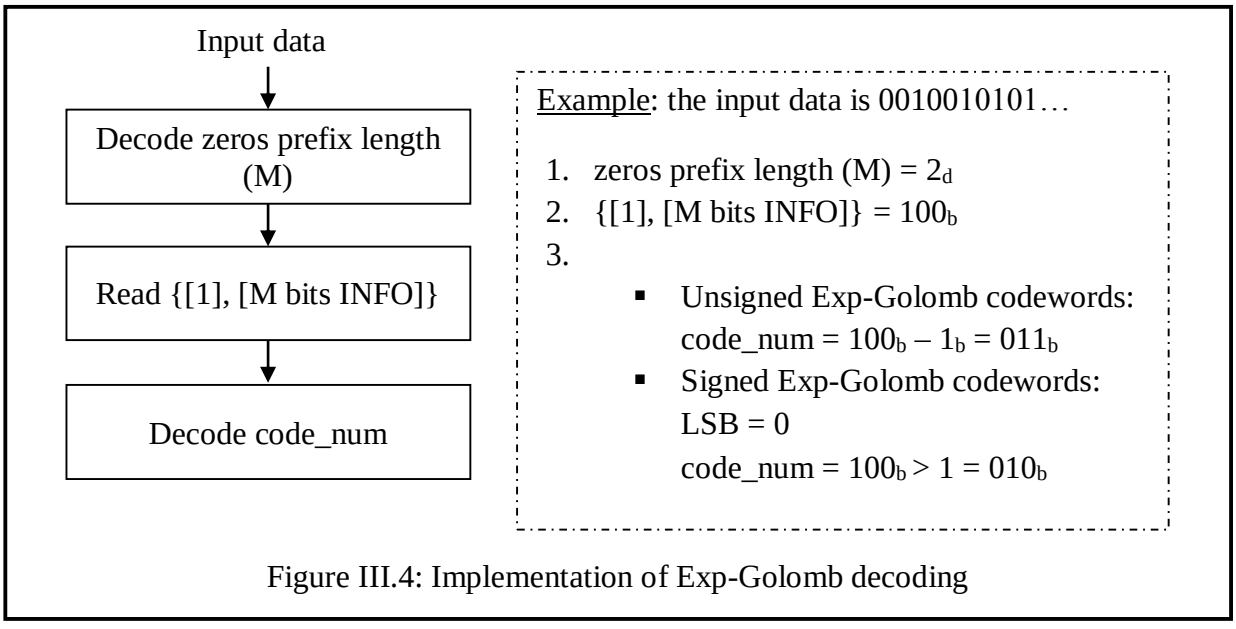


Figure III.4: Implementation of Exp-Golomb decoding

III.3.2 CAVLD decoding

In order to design the sub-module *CAVLD decoding*, the look-up table or ROM table is needed because many parameter in CAVLD decoding such as: *coeff_token*, *total zeros*, and *run before* used look-up table.

The look-up table is created by using the method table partitioning, called *Node-Leaf method*. Figure III.5 shows the *Node-Leaf method* algorithm.

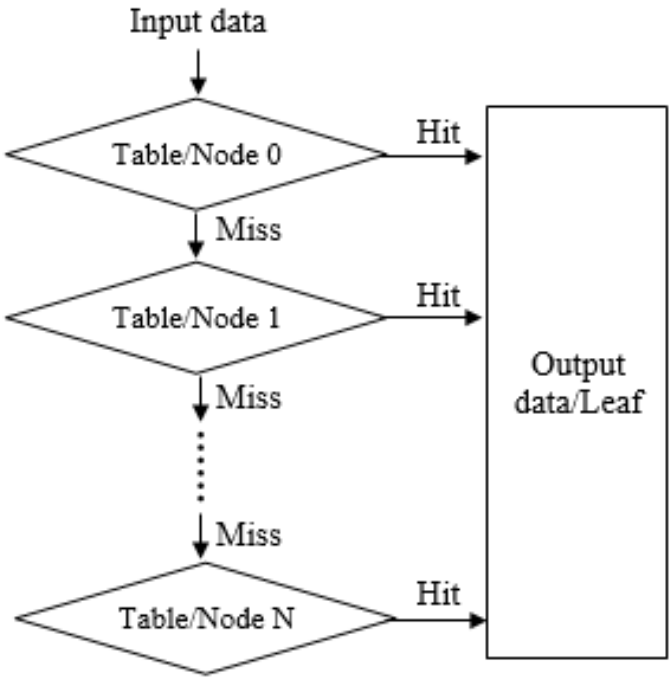


Figure III.5: *Node-Leaf method* algorithm

The *Node-Leaf method* algorithm with *N* bits partitioning processes as the following:

- Reading *N* bits of input data. These *N* bits are processed in the initial Table/Node (for example Table/Node 0). If these *N* bits are a node (not the LSB bits of input data), these *N* bits indicate the next look-up Table/Node (for example Table/Node K) and the process continues to read the next *N* bits of input data which are processed in the corresponding Table/Node (for example Table/Node K) addressed by the previous *N* bits of input data. Otherwise, it will output the corresponding data in the look-up Table/Node.

Example: With the input data is 00000011 and the look-up table of *total_zeros* shown in Figure III.6, the *Node-Leaf method* algorithm with 3 bits partitioning processes as follow:

1. 1st step: reading 3 bits of input data, we got 000 bits. These 3 bits (000 bits) are a node and it addresses the next look-up table or node: Node 1.
2. 2nd step: reading next 3 bits of input data, we got 000 bits which is looked up in the Table/Node 1. These 3 bits (000 bits) are a node and it addresses the next look-up table or node: Node 2.
3. 3rd step: reading next 3 bits of input data, we got 11X bits which is looked up in the Table/Node 2. These 3 bits (11X bits) are a leaf (not a node), so the output data with corresponding to these 3 bits (11X bits) are the value 11 in decimal.

Table 9-7 : total_zeros tables for 4x4 blocks with TotalCoeff(coeff_token) 1 to 7

TotalCoeff(coeff_token) = 1

total_zeros	Len	Bin	Bin with don't cares (9 bits width)
15	9	000000001	000000001
14	9	000000010	000000010
13	9	000000011	000000011
12	8	00000010	00000010X
11	8	00000011	00000011X
10	7	0000010	0000010XX
9	7	0000011	0000011XX
8	6	000010	000010XXX
7	6	000011	000011XXX
6	5	00010	00010XXXX
5	5	00011	00011XXXX
4	4	0010	0010XXXXX
3	4	0011	0011XXXXX
2	3	010	010XXXXXX
1	3	011	011XXXXXX
0	1	1	1XXXXXXXX

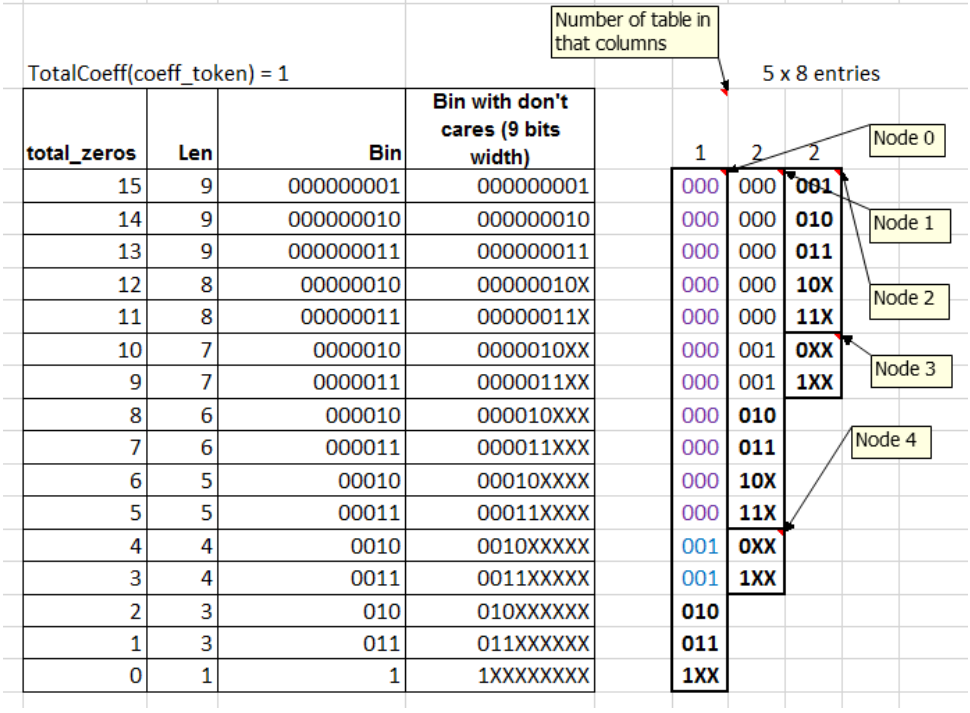


Figure III.6: Look-up table of *total_zeros*

The detail of the implementation of *CAVLD decoding* is provided in reference [6].

III.4 Results

The simulation results of the implementation is obtained by the simulation the design in ActiveHDL with the two modules of test bench added: *tb_clock module* to generate signal clock and reset; *tb_read_bitstream module* to read one byte from file (*file .264*) providing the video bitstream to the bitstream decoder module (*bsd module*).

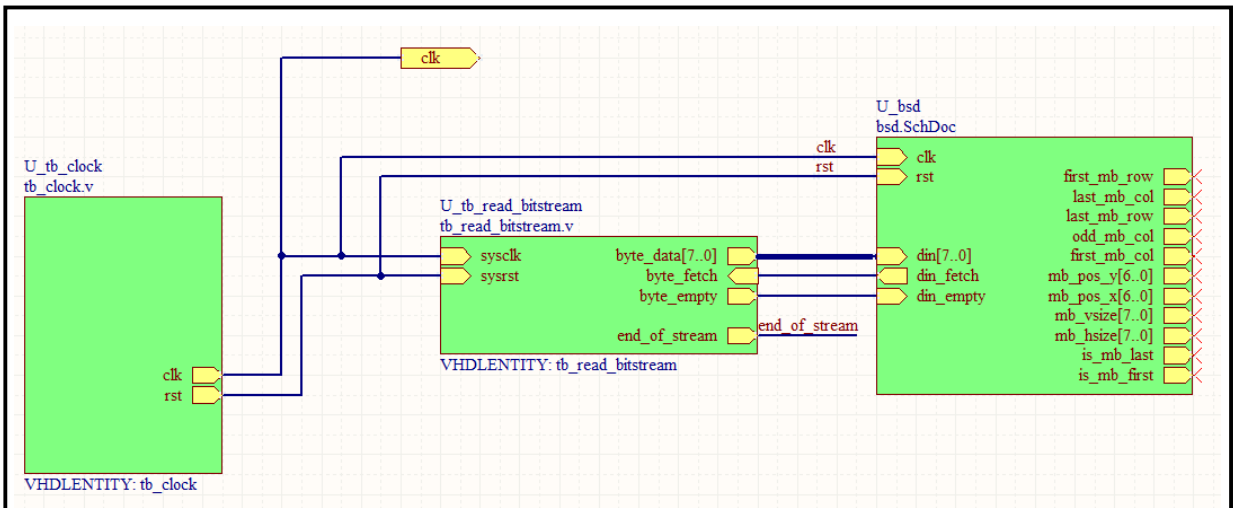


Figure III.7: Top-level test bench block schematics of bitstream decoder in Altium designer

The Figure III.8 shows the implementation in ActiveHDL. The results is verified by the simulation chronograms and the output file of the simulation *verilog_trace.txt*.

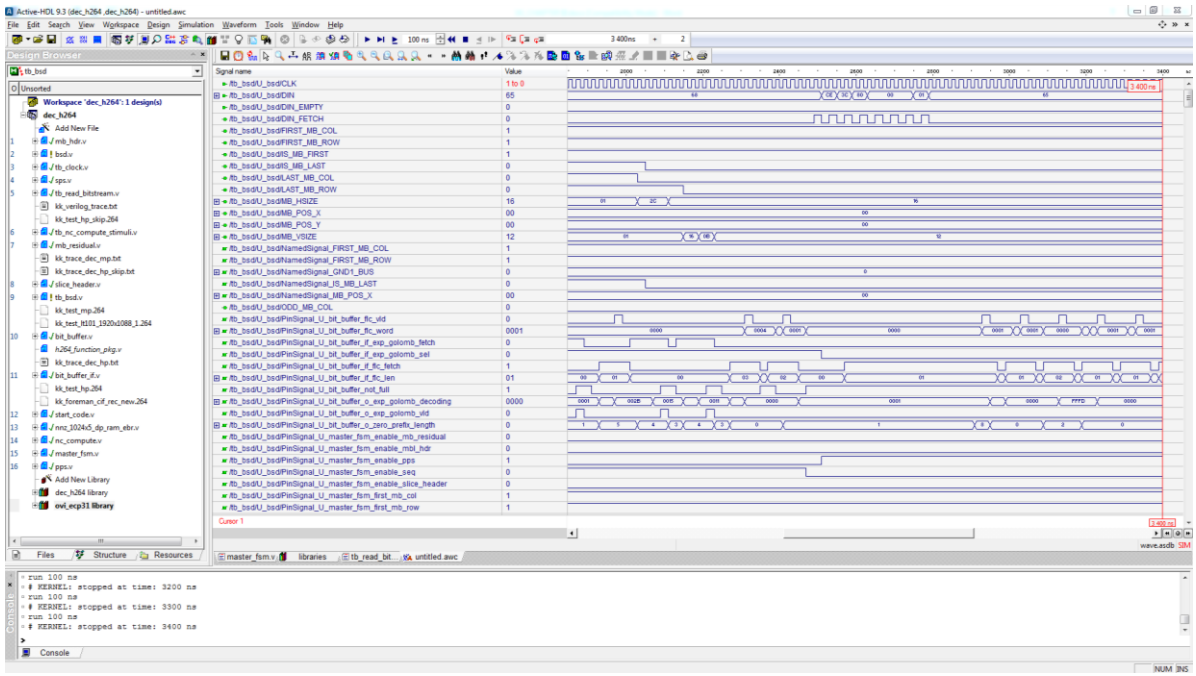


Figure III.8: Simulation results in ActiveHDL

The Figure III.9 shows the trace file created by the simulation in ActiveHDL and the trace file created by *Joint Model* (JM) ITU-T H.264 encoder and decoder C code model.

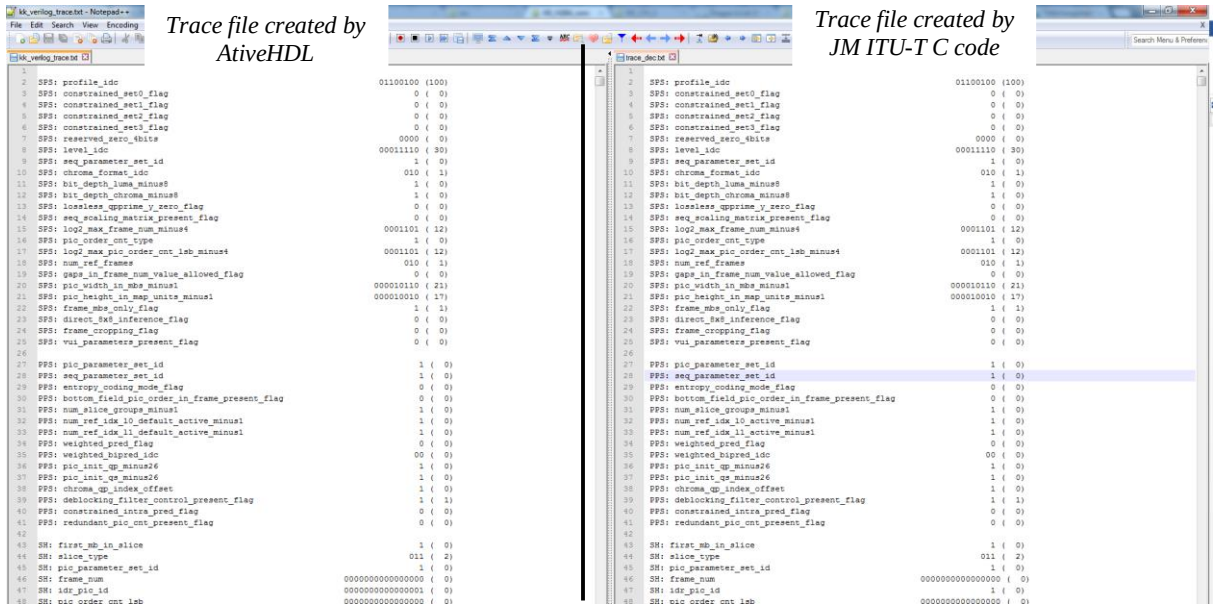


Figure III.9: Trace file created by the simulation in ActiveHDL and the trace file created by JM ITU-T C code model project

Without the *CAVLD decoding* and the *Macroblock data decode*, the synthesize design, map design, place and route design in *Diamond 3.1* on the Lattice FPGA device - LFE3-150EA-6FN1156C show that our design can run on the maximum frequency of 198.334 MHz with the clock constraint of 150 MHz. And, the resource FPGA used in the design is shown in Table III.1.

Table III.1: Resources FPGA used in the design

Number of registers:	468 out of 115296 (0 %)
Number of SLICEs:	535 out of 74520 (1 %)
Number of LUT4s:	852 out of 149040 (1 %)

The Figure III.10 shows the *CAVLD ROM* table file in *.mem* generated by the python source codes. The *IP* Express diamond will use this *.mem* file to generate the *CAVLD ROM* table in Verilog (*.v*) file, as shown in Figure III.11.

vlc_rom.mem

```
#total_coeff_n_trailing_ones_0  0,0 ( 0) -> [1]
00000000001
#total_coeff_n_trailing_ones_0  0,1 ( 0)         Idx  10
10000001010
#total_coeff_n_trailing_ones_0  0,2 ( 1)         Idx   5
10010000101
#total_coeff_n_trailing_ones_0  0,3 ( 1)         Idx   5
10010000101
#total_coeff_n_trailing_ones_0  0,4 ( 2)         Idx   0
10100000000
#total_coeff_n_trailing_ones_0  0,5 ( 2)         Idx   0
10100000000
#total_coeff_n_trailing_ones_0  0,6 ( 2)         Idx   0
10100000000
#total_coeff_n_trailing_ones_0  0,7 ( 2)         Idx   0
10100000000
#total_coeff_n_trailing_ones_0  1,0 ( 0) -> [2]
00000000010
#total_coeff_n_trailing_ones_0  1,1 ( 0) -> [21]
00000010101
#total_coeff_n_trailing_ones_0  1,2 ( 0) -> [22]
00000010110
#total_coeff_n_trailing_ones_0  1,3 ( 0)         Idx  19
10000010011
#total_coeff_n_trailing_ones_0  1,4 ( 0)         Idx   9
10000001001
#total_coeff_n_trailing_ones_0  1,5 ( 0)         Idx   4
10000000100
#total_coeff_n_trailing_ones_0  1,6 ( 1)         Idx  15
10010001111
```

Figure III.10: *CAVLD ROM* table file in *.mem* generated by the python source codes

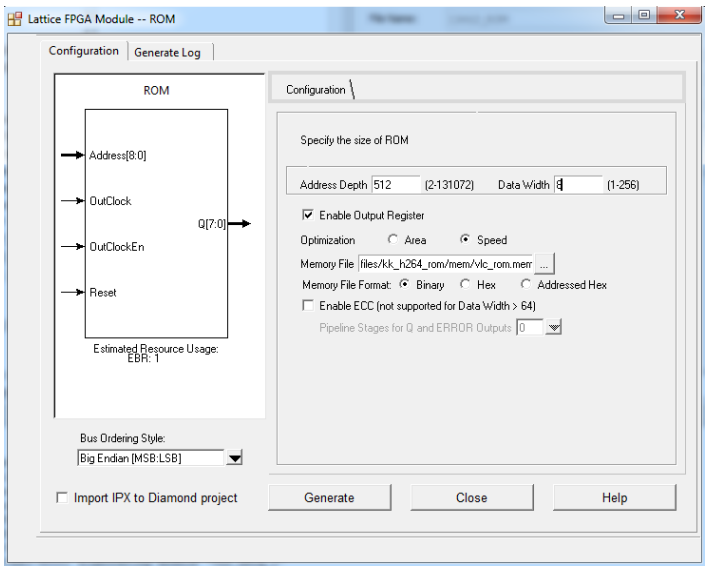


Figure III.11: Lattice *FPGA* module – *ROM* in *IP* Express diamond

CONCLUSION

In this master thesis, chapter I described the first part of my work during the internship. It was devoted to a description of the SDI video interface and the demonstration of the implementation of a Lattice tri-rate SDI PHY IP core on a Lattice FPGA embedded in the Enciris LT-125 board. Chapter I provides a quite large amount of information about the high data-rate SERDES/PCS interface of Lattice ECP3 FPGAs, the Lattice tri-rate SDI PHY IP core, and the Enciris LT-125 board. The most interesting aspect in that task was due to the complexity of both the SERDES/PCS interface and the Lattice tri-rate SDI PHY IP core: configuration and generation of the related IP cores were really subtle and sometimes mind-boggling. However, the Lattice SDI IP core was successfully implemented on the Enciris LT-125 board, which now possesses an SMPTE 3G/HD/SD SDI video input functionality. This met the technical needs of the company who had not yet had any boards which support an SDI video input functionality by incorporating with the Lattice SDI IP core.

The second part of my internship is focused on the hardware architecture design and implementation of an H.264/AVC bitstream decoder (BSD) which was specifically dealt with in the subsequent chapters. The CAVLD decoder was actually the most challenging module. I successfully managed to design the bitstream decoder which can handle the decoding of SPS, PPS, Slice header, and Macroblock header layers of the bitstream, as well as the CAVLD decoding of some 4x4-block fields by using the VLC tree table method (*Node-Leaf method*). The preliminary synthesis of the design yields a maximum operating frequency of 198.334 MHz on a Lattice LFE3-150EA-6FN1156C FPGA device with a clock frequency constraint of 150 MHz.

In the near future, the bitstream decoder will be completed by adding the finalized CAVLD decoding sub-module, and the macroblock data decoder module. Afterwards, the IT/IQ, intra prediction and inter prediction modules will need to be designed and integrated in order to form the H.264/AVC decoder. The purpose is to develop a commercial high data-rate (about 50-100 Mbps) H.264 1080p-P30 decoder compliant with the Constrained Baseline Profile, and level 4.

Throughout my internship, several technical and scientific aspects were covered:
First of all, I discovered and dug into the amazing world of advanced video coding/decoding with its lengthy standards and elaborate video processing algorithms. Second, I have gone through an entire FPGA development cycle (HDL code design, functional simulation, testing, and full-scale validation) which helped me understand and practice the full scope and the innumerable difficulties of FPGA hardware design, development and implementation, in a real-life industrial context. Hence, I was naturally led to use a variety of software tools for FPGA design such as: Diamond, IPexpress, Altium Designer, Python, etc. It is worthy to note that due to the Enciris statement of work I was also forced to learn from scratch the Verilog HDL modelling and programming language, which I find offers more flexibility and potentialities than VHDL. Finally, my work allowed me to better grasp the stakes of embedded electronic system development with highly constrained operating and computational performance specifications. It was a great experience: I much enjoyed the work of an R&D electrical engineer, the supreme reward being the satisfaction of seeing my design work in a full scale commercial product, which supplies the bridge between theoretical training and industrial achievements. I am grateful to Enciris for having accepted to let me participate in this great technological and personal venture.

REFERENCES

[1] LatticeECP3 SERDES_PCS Usage Guide.pdf

[2] Tri-Rate Serial Digital Interface Physical Layer IP Core User's Guide.pdf

[3] Model LT-125 USER MANUAL.pdf

[4] Schematic_LT-125-REV0-ASSY0.pdf

[5] IAIN E. Richardson. H.264 and MPEG-4 Video Compression. John Wily and Sons Ltd., England, 2003.

[6] ITU-T (2005). ITU-T Recommendation H.264- H.264 telecommunication standardization sector of itu (03/2005) series h: audiovisual and multimedia systems Infrastructure of audiovisual services – Coding of moving video Advanced video coding for generic audiovisual services

[7] *Alexsandro C. Bonatto, Henrique A. Klein, Marcelo Negreiros, André B. Soares, Letícia V. Guimarães and Altamiro A. Susin*. Hardware Decoding Architecture for H.264/AVC Digital Video Standard. Department of Electrical Engineering Federal University of Rio Grande do Sul, Brazil.

[8] *Ke Xu*. nova: a H.264/AVC Baseline Decoder Specification. OpenCores.Org, 2014.

[9] JM 18.6 (2014). H.264/AVC Software Coordination. Available at: http://iphome.hhi.de/suehring/tml/download/

Student name:	Mr. Vicheka PHOR
Department:	CAMSI
Academic year:	2013-2014
Company:	Enciris Technologies Company
Title of internship:	"Video decoding: SDI interface implementation & H.264/AVC bitstream decoder hardware architecture design and implementation"
Internship duration:	5 months and a half

REPORT SUMMARY

Nowadays, digital video is widely used in many applications in the way of creating or sharing for example the digital television broadcasting, internet video streaming, mobile video streaming, DVD video and video calling application. Therefore, effective video coding is an essential component of these applications and can make the difference between the success and failure of a business model. The video coding is the process of compressing and decompressing a digital video signal which is sent/received by various interfaces such as: SD-SDI, HD-SDI, 3G-SDI, DVI and HDMI. There are many methods or algorithms to compress and decompress digital video such as: VC1, H.262/MPEG-2, H.263/MPEG-4, and H.264/AVC algorithm. Among these video coding algorithms, H.264/AVC has huge significance to the broadcast, internet, consumer electronics, mobile and security industries application. Also, H.264/AVC is the latest in a series of standards published by the ITU and ISO. It describes and defines a method of coding video that can give better performance than any of the preceding standards. H.264/AVC makes it possible to compress video into a smaller space, which means that a compressed video clip takes up less transmission bandwidth and/or less storage space compared to older codecs.

This master thesis was organized in two parts:

- The first part which is composed of one chapter: Chapter I, studied about the SDI video interface by demonstrating the implementation of a Lattice tri-rate SDI PHY IP core on a Lattice FPGA of the Enciris LT-125 board in order to provide an SMPTE 3G/HD/SD SDI video input functionality on the board without using any external chipset like the Gennum chipset which is used today on Enciris boards.

- The second part which contains two chapters: Chapter II and Chapter III, described about the H.264/AVC video decoding algorithm, particularly the H.264/AVC bitstream decoder hardware architecture design and implementation.

In the first part of this master thesis, I have successfully implemented a Lattice tri-rate SDI PHY IP core on the Enciris LT-125 board, provided an SMPTE 3G/HD/SD SDI video input functionality on the board that met the technical needs of the company. As for the second part, the implementation of bitstream decoder is not totally completed yet because of lacking time and the complexity of CAVLD decoding. Somehow, I successfully managed to design the bitstream decoder which can handle the decoding of SPS, PPS, Slice header, and Macroblock header layers of the bitstream, as well as the CAVLD decoding of some 4x4-block fields by using the VLC tree table method (*Node-Leaf method*). The preliminary synthesis of the design yields a maximum operating frequency of 198.334 MHz on a Lattice LFE3-150EA-6FN1156C FPGA device with a clock frequency constraint of 150 MHz.