2010

Algorithms & implementation of advanced video coding standards

Jianjun Li
University of Windsor

Algorithms & Implementation of Advanced Video Coding Standards

by

Jianjun Li

A Dissertation
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfilment of the Requirements for
the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada

2010

Your file Votre référence
ISBN: 978-0-494-62760-0
Our file Notre référence
ISBN: 978-0-494-62760-0

Abstract

Advanced video coding standards have become widely deployed coding techniques used in numerous products, such as broadcast, video conference, mobile television and blu-ray disc, etc. New compression techniques are gradually included in video coding standards so that a 50% compression rate reduction is achievable every five years. However, the trend also has brought many problems, such as, dramatically increased computational complexity, co-existing multiple standards and gradually increased development time. To solve the above problems, this thesis intends to investigate efficient algorithms for the latest video coding standard, H.264/AVC. Two aspects of H.264/AVC standard are inspected in this thesis: (1) Speeding up intra4x4 prediction with parallel architecture. (2) Applying an efficient rate control algorithm based on deviation measure to intra frame. Another aim of this thesis is to work on low-complexity algorithms for MPEG-2 to H.264/AVC transcoder. Three main mapping algorithms and a computational complexity reduction algorithm are focused by this thesis: motion vector mapping, block mapping, field-frame mapping and efficient modes ranking algorithms. Finally, a new video coding framework methodology to reduce development time is examined. This thesis explores the implementation of MPEG-4 simple profile with the RVC framework. A key technique of automatically generating variable length decoder table is solved in this thesis. Moreover, another important video coding standard, DV/DVCPRO, is further modeled by RVC framework. Consequently, besides the available MPEG-4 simple profile and China audio/video standard, a new member is therefore added into the RVC framework family.

A part of the research work presented in this thesis is targeted algorithms and implementation of video coding standards. In the wide topic, three main problems are investigated. The results show that the methodologies presented in this thesis are efficient and encouraged.

Acknowledgements

The research that has gone into this thesis has been thoroughly enjoyable. That enjoyment is largely a result of the interaction that I have had with my supervisors, committee members and colleagues.

I feel very privileged to have worked with my supervisor, Dr. Esam Abdel-Raheem. I would like to thank him for sharing his sorrows and happiness with me. We spent long hours together in writing papers. It is remarkable that we have worked together for as long as 12 hours a day. I am grateful to him for his moral support during my difficult days of my life.

I am very grateful to my committee members Dr. Mahmoud El-Sakka, Dr. Huapeng Wu, Dr. Mohmmad Khalid and Dr. Bubaker Boufama. To each of them I owe a great debt of gratitude for their patience, inspiration and participation in this work. I would also like to thank our graduate student chair Dr. Xiang Chen for giving me an opportunity to become a proud student of this university. I would also like to thank Dr. Maher Sid-Ahmed for always asking me to work hard to graduate as soon as possible. I spent many enjoyable hours with department members and fellow students chatting about my latest crazy ideas over a cup of coffee. Without this rich environment I doubt that many of my ideas would have come to be real.

Thanks also to my brother who has been extremely understanding and supportive of my studies. I feel very lucky to have a wonderful family. They share my enthusiasm for academic pursuits.

Windsor, April 2010

Jianjun Li

Contents

List of Figures

List of Tables

List of Abbreviations

3GPP	3rd Generation Partnership Project
AAUX	Audio Auxiliary Information
AS	AAUX Source
ASC	AAUX Source Control
ASO	Arbitrary Slice Ordering
AVC	Advanced Video Coding
B	Bidirectional Frame
BG	Binary Group
BP	H.264/AVC Baseline Profile
BPP	Bit Rate Per Picture
BSDL	Bitstream Syntax Description Language
CAL	Caltrop Actor Language
CABAC	Context-Adaptive Binary Arithmetic Coding
CAVLC	Context-Adaptive Variable-Length Coding
CBR	Constant Bit Rate
CBP	Coded Block Pattern
DCT	Digital Cosine Transfer
DDL	Decoded Description Language
DIF	Digital Interface
DV	Digital Video
DVC	Distributed Video Coding Standard

DWA	Distance Weighted Average
EI	Entropy Information
EWA	Error-variance Weighted Average
FCBR	Frame based Constant Bit Rate
FMO	Flexible Macroblock Ordering
FSM	Finite State Machine
FU	Function Units
GOP	Group of Picture
H.264/AVC	Advanced Video Coding standard
HD	High Definition
HD-TV	High Definition TV
IDCT	Inverse Digital Cosine Transfer
I	Intra Frame
IM	Intra16 DC Mode
IQ	Inverse Quantization
IRSL	Image and Remote Sensing Laboratory
ITU	International Telecommunication Union
JVT	Joint Video Team
LOC	Lines Of Code
LoG	Laplacian of Gaussian
MAD	Mean Absolute Different
MB	Macro Block
MBAFF	MacroBlock Adaptive Frame/Field
MC	Motion Compression
ME	Motion Estimation
MP	H.264/AVC Main Profile
MPEG	ISO/IEC Moving Picture Experts Group
MPEG-4 SP	MPEG-4 Simple Profile

MRF	Multiple Reference Frames
MSE	Mean Square Error
MV	Motion Vector
MVC	Multiview Video Coding Standard
NAL	Network Adaptation
NTSC	National Television System Committee
P	Predicted Frame
PAL	Phase Alternating Line
PDA	Personal Digital Assistant
PSNR	Peak Signal to Noise Ratio
Q	Quantization
QP	Quatization Parameter
QSMC	Quarter Sub-pixel Mmotion Compensation
R-Q	Rate-Quantization
RC	Rate Control
RDO	Rate-Distortion Optimized
RS	Redundant Slices
RTL	Register Transfer Level
RVC	Reconfigurable Video Coding
SAD	Sum of Absolute Different
SD	Standard Definition
SI	switching intra
SP	switching predictive
SSD	Sum of Squared Distortion
STAD	Sum of Transformation Absolute Distortion
SVC	Scalable Video Coding Standard, the scalable extension of H.264/AVC
TC	Time Code
UVLC	Universal Variable Length Coding

VAUX	Video Auxiliary Information
VBR	Variable Bit Rate
VBS	Variable Block Size
VCEG	Video Coding Experts Group
VCEG	ITU-T Video Coding Expert Group
VCL	Video Coding Layer
VLC	Variable Length Coding
VLD	Variable Length Decoding
XML	Extensible Markup Language
VS	VAUX Source
VSC	VAUX Source Control
XP	H.264/AVC Extended Profile
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations
VTL	Video Coding Tools Library

Chapter 1

Introduction

Digital multimedia systems, such as digital television and video streaming over the Internet, belong to the everyday life of many people as shown in Fig. 1.1. Due to the fact that uncompressed video requires a huge bandwidth, the input video is compressed by the source coder to a desired bit rate. The encoder performs lossy video signal compression. Then, the data is transmitted to the receiver side via a transmission channel. The receiver performs inverse operations to obtain a reconstructed video signal for display.

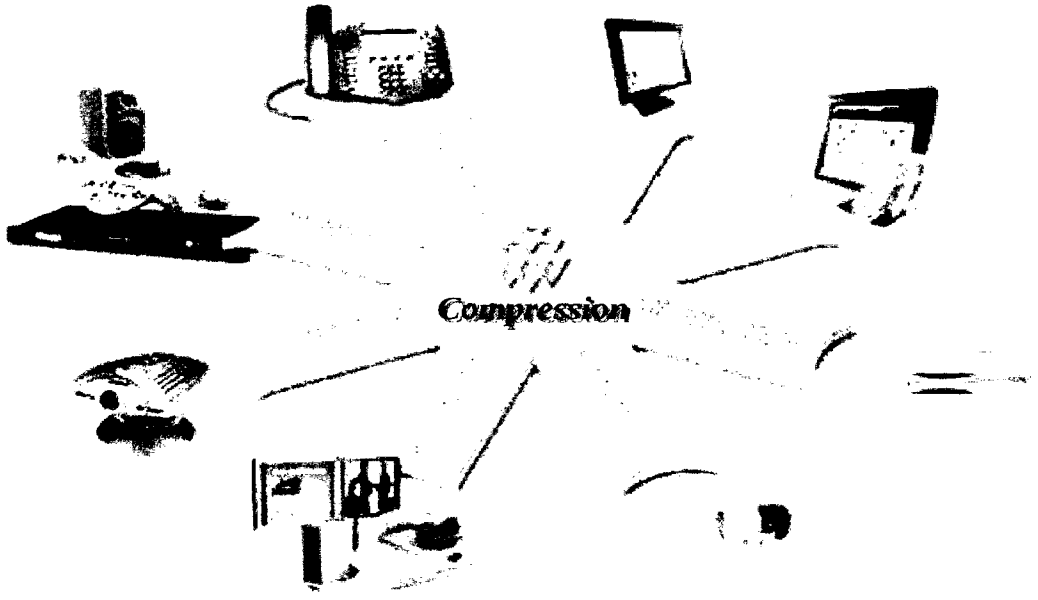


Figure 1.1: Multimedia systems.

Continuous emergence of video coding standards on one side, and the growth in development and implementation technology for them on the other side, have undoubtedly created a whole new world of multimedia. So far, contributions to video coding technology have mainly focused on improving coding efficiency. The challenges remain: not only to find efficient coding algorithms which require both simplification and high performance but also to reduce the design time and avoid repeating design. This thesis starts with finding efficient algorithms for the latest video coding standard, H.264/AVC, and then moves on a transition methodology from the most prevalent video coding standard, MPEG-2, to the most efficient video coding standard, H.264/AVC. Finally, a reconfigurable video coding framework is presented to reduce development time.

1.1 Motivations & Contributions

Video coding standard is a large scope and it is impossible for a thesis work to overcome all the issues. In this thesis, three major problems are emphasized: Low-complexity and efficient H.264/AVC algorithms, MPEG-2/H.264 transcoding methodology and reconfigurable video coding framework implementation.

At first, the latest video coding standard, H.264/AVC, is getting more attention due to its high compression efficiency. However, higher computational complexity has to be paid for the advantage. Being four times higher computational complexity than its counterpart, MPEG-2, is considered an obstacle to implement it in real-time. Therefore, many research works focus on how to reduce the computational complexity of H.264/AVC. The intra4x4 prediction is main contribution of H.264/AVC. The available research results do not make full use of the features of the intra4x4 prediction so that processing time is increased and is not suitable for real-time process. The thesis proposes a new parallel processing structure to reduce the processing time by carefully analysing the feature of H.264/AVC intra4x4 prediction [1]. On the other hand, rate control is playing a crucial role with limited bandwidth. How to improve video quality in constant bit rate (CBR) is a great concern. In this thesis, a deviation based rate control algorithm reasonably solves this problem for intra frame [2]. Two new encoder schemes are also proposed in this thesis.

Secondly, multiple standards co-exist is also an important issue because the older standard

is still used prevalently when the newer standard emerges. Apparently, to deal with this problem, the transcoding algorithms are required. A universal transcoder, which can transcode between any video formats, is not realistic. In this thesis, the methodology to transcode the most prevalent video format, MPEG-2, to the latest video format, H.264/AVC, is presented [3–5].

Finally, multiple codec standards need to be supported because more and more video standards are deployed. Although different, all coding standards use the same or very similar coding tools and hence share similar architectures and implementations. Unfortunately, the way in which the existing coding standards are specified lacks flexibility to adapt performances and complexity when new applications emerge. Therefore, repeating design and long development time are imperative. A framework methodology using tools library has been proposed by MPEG organization. The reconfigurable video coding (RVC) standard intends to create a framework containing existing coding technology for developing, beside current standard decoders, new configurations for satisfying specific application constraints. However, some issues still exist in that it is a brand new standard, such as, how to separate the variable length decoding (VLD) from the decoder parser unit? how to implement a new video coding standard with RVC framework, and how to utilize the available function units (FUs) in design? the author contributes an efficient data flow based implementation of the variable length decoding (VLD) process particularly adapted for the instantiation and synthesis of CAL parses in the MPEG RVC framework in the paper [6]. Three contributions [7–9] have been adopted by MPEG RVC CAL reference code. The research work also models DV/DVCPRO video coding standard with the MPEG RVC framework [10].

1.2 Thesis Organization

The research work presented in this thesis is categorized into four parts as shown in Fig. 1.2. There are eight chapters included in this thesis. Chapter 1 introduces the motivation of this thesis. The video coding standards are reviewed in Chapter 2, the focus being on those features that are relevant for the thesis. The second part of this thesis presents how to reduce the computational complexity and implement efficient rate control process for H.264/AVC. Chapter 3 and 4 are included in this part. Chapter 3 focuses on a fast intra4x4 prediction

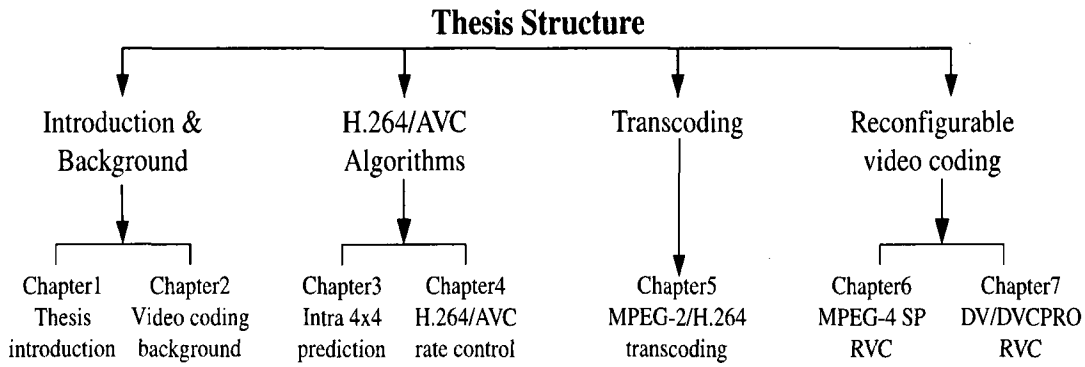


Figure 1.2: Thesis structure.

process methodology. Chapter 4 deals with an efficient rate control algorithm of H.264/AVC. The rate control algorithm is the most important part in H.264/AVC standard, particularly in today, when the video is transmitted on net with limited bandwidth. The third part implements MPEG-2 to H.264/AVC transcoding. The main issues and methodologies of transcoding are presented in Chapter 5. The fourth part launches a new video coding standard, a framework video coding process, which is being developed by MPEG organization. In this part, Chapter 6 presents novel methodology to automatically generating variable length decoding table for RVC. Chapter 7 successfully implements DV/DVCPRO video coding standard with RVC-CAL. In the end, Chapter 8 provides a summary of this work. Some future research directions have been proposed in the same chapter.

Chapter 2

Background

In this chapter, some background knowledge about video compression is provided. A more complete discussion on this subject can be found in books [11–15] and specifications [16–21], such as, video compression methods, video coding standards, profile, level, motion vector, macroblock, and peak signal-to-noise ratio (PSNR).

2.1 Video Coding Compression Principle

2.1.1 Exploiting Temporal Redundancies

Since a video is essentially a sequence of pictures sampled at a discrete frame rate, two successive frames in a video sequence look largely similar. The extent of similarity between two successive frames depends on how closely they are sampled (frame interval) and the motion of the objects in the scene. Exploiting the temporal redundancies accounts for majority of the compression gains in video encoding.

Since two successive frames are similar, taking the difference between the two frames results in a smaller amount of data to be encoded. In general, the video coding technique that uses the data from a previously coded frame to predict the current frame is called predictive coding technique. The computation of the prediction is the key to efficient video compression. The simplest form of predictive coding is frame difference coding, where, the previous frame is used as a prediction. The difference between the current frame and the predicted frame is then encoded. The frame difference prediction begins to fail as the object motion in a video

sequence increases resulting in a loss of correlation between collocated pixels in two successive frames.

Object motion is common in video and even a small motion of 1 to 2 pixels can lead to loss of correlation between corresponding pixels in successive frames. Motion compensation (MC) is used in video compression to reduce the correlation lost due to object motion [11]. The object motion in the real world is complex but for the purpose of video compression, a simple translational motion is assumed.

2.1.1.1 Block Based Motion Estimation

If we observe two successive frames of a video, the amount of changes within small NxN pixel regions of an image are small. Assuming a translational motion, the NxN regions can be better predicted from a previous frame by displacing the NxN region in the previous image by an amount representing the object motion. The amount of this displacement depends on relative motion between the two frames. For example, if there is a 5 pixel horizontal motion between the frames, it is likely that a small NxN region will have a better prediction if the prediction comes from a NxN block in the previous image displaced by 5 pixels. The process of finding a predicted block that minimizes the difference between the original and predicted blocks is called motion estimation (ME) and the resulting relative displacement is called a motion vector [11]. When motion compensation is applied to the prediction, the motion vector is also coded along with the pixel differences.

Video frames are typically coded one block at a time to take advantage of the motion compensation applied to small NxN blocks. As the block size decreases, the amount of changes within a block also typically decrease and the likelihood of finding a better prediction improves. Similarly, as the block size increases, the prediction accuracy decreases. The downside to using a smaller block size is that the total number of blocks in an image increases. Since each of the blocks also has to include a motion vector to indicate the relative displacement, the amount of motion vector information increases for smaller block sizes.

The best prediction for a given block can be found if the motion of the block relative to a reference picture can be determined. Since translational motion is assumed, the estimated motion is given in terms of the relative displacement of a block in the X and Y planes. The

process of forming a prediction thus requires estimating the relative motion of a given NxN block. A simple approach to estimating the motion is to consider all possible displacements in a reference picture and determine which of these displacements gives the best prediction. The best prediction will be very similar to the original block and is usually determined using a metric such the minimum sum of absolute differences (SAD) of pixels or the minimum sum of squared differences (SSD) of pixels. The SAD has lower computational complexity compared to the SSD computation and equally good in estimating the best prediction. The number of possible displacements (motion vectors) of a given block is a function of the maximum displacement allowed for motion estimation. Fast motion estimation (FME) [22] has been an active area of research and a number of efficient algorithms have been developed.

2.1.1.2 Block Based Motion Compensation

Motion compensation describes a picture in terms of the transformation of a reference picture to the current picture. The reference picture may be previous in time or even from the future. When images can be accurately synthesized from previously transmitted/stored images, the compression efficiency can be improved. Motion compensation exploits the fact that, often, for many frames of a movie, the only difference between one frame and another is the result of either the camera moving or an object in the frame moving. In reference to a video file, this means much of the information that represents one frame will be the same as the information used in the next frame. Motion compensation takes advantage of this to provide a way to create frames of a movie from a reference frame [12].

In block motion compensation (BMC), the frames are partitioned in blocks of pixels (e.g. macroblocks of 16x16 pixels in MPEG). Each block is predicted from a block of equal size in the reference frame. The blocks are not transformed in any way apart from being shifted to the position of the predicted block. This shift is represented by a motion vector.

To exploit the redundancy between neighboring block vectors, (e.g. for a single moving object covered by multiple blocks) it is common to encode only the difference between the current and previous motion vector in the bit stream. The result of this differencing process is mathematically equivalent to a global motion compensation capable of panning. Further down the encoding pipeline, an entropy coder will take advantage of the resulting statistical

distribution of the motion vectors around the zero vector to reduce the output size.

2.1.2 Exploiting Spatial Redundancies

In natural images, there exists a significant correlation between neighboring pixels. Small areas within a picture are usually similar. Redundancies exist even after motion compensation. Exploiting these redundancies will reduce the amount of information to be coded. Prediction based on neighboring pixels, called intra prediction [11], is also used to reduce the spatial redundancies. Transform techniques are used to reduce the spatial redundancies substantially. The spatial redundancy exploiting transforms such as the discrete cosine transform (DCT), transform a NxN picture block into NxN block of coefficients in another domain called the frequency domain [23]. The key properties of these transforms that make them suitable for video compression are energy compaction and de-correlation. When the transform is applied, the energy of a NxN pixel block is compacted into a few transformed coefficients and the correlation between the transformed coefficients is also reduced substantially. This implies that significant amount of information can be recovered by using just a few coefficients. The transform coefficients in the frequency domain can be roughly classified into low, medium, and high spatial frequencies [13]. Since the human visual system is not sensitive to the high spatial frequencies, the transform coefficients corresponding to the high frequencies can be discarded without affecting the perceptual quality of the reconstructed image. As the number of discarded coefficients increases, the compression increases, and the video quality decreases. The coefficient dropping is in fact exploiting the perceptual redundancies. Another way of reducing the perceptual redundancies is by quantizing the transform coefficients. The quantization process reduces the number of levels [21] while still retaining the video quality. As with coefficient dropping, as the quantization step size increases, the compression increases, and the video quality decreases.

2.1.3 Exploiting Statistical Redundancies

The transform coefficients, motion vectors, and other data have to be encoded using binary codes in the last stage of video compression. The simplest way to code these values is by using fixed length codes, e.g. 16 bit words. However, these values do not have a uniform distribution

and using fixed length codes is wasteful. Average code length can be reduced by assigning shorter code words to values with higher probability. Variable length coding is used to exploit these statistical redundancies and increase compression efficiency further.

2.2 Image/Video Coding Roadmap

The multimedia compression standards have been developing for decades as shown in Fig.2.1. The Moorse's law [24] of compression shows that the performance has been doubled every five years, which means the standard is able to obtain roughly 50% gain in about five years. In the mean time, the computational complexity also increases dramatically.

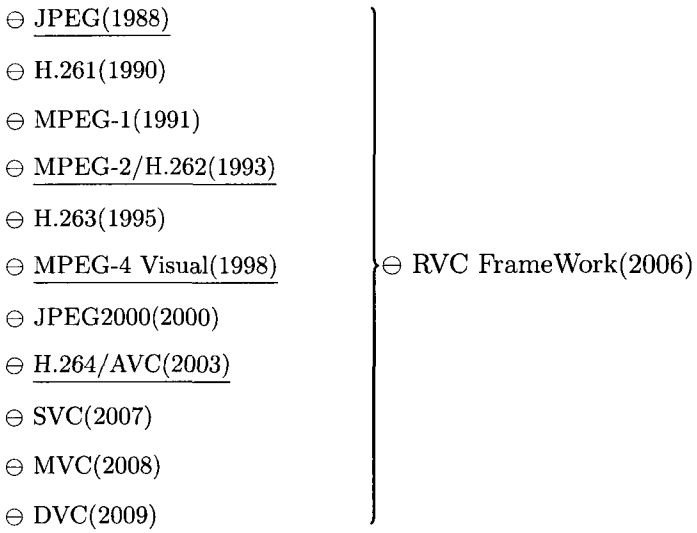


Figure 2.1: Image/Video coding standards development roadmap.

2.3 Video Coding Standards Overview

2.3.1 H.261/H.263

The H.261 [17] and H.263 [19] are not international standards but only recommendations of the ITU. They are both based on the same technique as the MPEG standards and can be seen as simplified versions of MPEG video compression. They were originally designed for video conferencing over telephone lines, i.e. low bandwidth. However, it is a bit contradictory that they lack some of the more advanced MPEG techniques to really provide efficient bandwidth

use. The conclusion is therefore that H.261 and H.263 are not suitable for usage in general digital video coding.

2.3.2 MPEG-1

The first public standard of the MPEG committee was the MPEG-1, ISO/IEC 11172 [16], whose first parts were released in 1993. MPEG-1 video compression is based upon the same technique that is used in JPEG. In addition to that, it also includes techniques for efficient coding of a video sequence. In Motion JPEG/Motion JPEG 2000, each picture in the sequence is coded as a separate unique picture resulting in the same sequence as the original one. In MPEG video, only the new parts of the video sequence is included together with information of the moving parts. MPEG-1 is focused on bit streams of about 1.5 Mbps and originally for storage of digital video on CDs. The focus is on compression ratio rather than picture quality. It can be considered as traditional VCR quality but digital instead. It is important to note that the MPEG-1 standard, as well as MPEG-2, MPEG-4 and H.264 that are described below, defines the syntax of an encoded video stream together with the method of decoding this bit stream. Thus, only the decoder is actually standardized. A MPEG encoder can be implemented in different way and a vendor may choose to implement only a subset of the syntax, providing it provides a bit stream that is compliant with the standard. This allows for optimization of the technology and for reducing complexity in implementations. However, it also means that there are no guarantees for quality – different vendors implement MPEG encoders that produce video streams that differ in quality.

2.3.3 MPEG-2

The MPEG-2 project focused on extending the compression technique of MPEG-1 to cover larger pictures and higher quality at the expense of a higher bandwidth usage. MPEG-2, ISO/IEC 13818 [18], also provides more advanced techniques to enhance the video quality at the same bit rate. The expense is the need for far more complex equipment. As a note, DVD movies are compressed using the techniques of MPEG-2.

2.3.4 MPEG-3

The next version of the MPEG standard, MPEG-3 was designed to handle HDTV, however, it was discovered that the MPEG-2 standard could be slightly modified and then achieve the same results as the planned MPEG-3 standard. Consequently, the work on MPEG-3 was discontinued.

2.3.5 MPEG-4

The next generation of MPEG, MPEG-4, is based upon the same technique as MPEG-1 and MPEG-2. Once again, the new standard focused on new applications. The most important new features of MPEG-4, ISO/IEC 14496 [20] and concerning video compression are the support of even lower bandwidth consuming applications, e.g. mobile devices like cell phones, and on the other hand applications with extremely high quality and almost unlimited bandwidth. In general the MPEG-4 standard is wider than the previous standards. It also allows for any frame rate, while MPEG-2 was locked to 25 frames per second in PAL and 30 frames per second in national television system committee (NTSC). When "MPEG-4" is mentioned in surveillance applications today it is usually MPEG-4 part 2 that is referred to. This is the "classic" MPEG-4 video streaming standard, a.k.a. MPEG-4 Visual. Some network video streaming systems specify support for "MPEG-4 short header", which is a H.263 video stream encapsulated with MPEG-4 video stream headers. MPEG-4 short header does not take advantage of any of the additional tools specified in the MPEG-4 standard, which gives a lower quality level than both MPEG-2 and MPEG-4 at a given bit rate.

2.3.6 H.264/AVC

H.264/AVC is the latest generation standard for video encoding. This initiative has many goals. It should provide good video quality at substantially lower bit rates than previous standards and with better error robustness – or better video quality at an unchanged bit rate. The standard is further designed to give lower latency as well as better quality for higher latency. In addition, all these improvements compared to previous standards were to come without increasing the complexity of design. An additional goal was to provide enough flexibility to

allow the standard to be applied to a wide variety of applications: for both low and high bit rates, for low and high resolution video, and with high and low demands on latency. The following three profiles were defined in the original standard, and remain unchanged in the latest version:

- Baseline Profile (BP): Primarily for low-cost applications that require additional data loss robustness, this profile is used in some videoconferencing and mobile applications. This profile includes all features that are supported in the Constrained Baseline Profile, plus three additional features that can be used for loss robustness (or for other purposes such as low-delay multi-point video stream composition).

- Extended Profile (XP): This profile is used for standard-definition digital TV broadcasts that use the MPEG-4 format as defined in the DVB standard [25]. It is not, however, used for high-definition television broadcasts, as the importance of this profile faded when the High Profile was developed in 2004 for that application.

- Main Profile (MP): Intended as the streaming video profile, this profile has relatively high compression capability and some extra tricks for robustness to data losses and server stream switching.

Table 2.1 gives a high-level summary of the coding tools included in these profiles. The baseline profile includes intra (I) and predictive (P)-slices, some enhanced error resilience tools (flexible macroblock ordering (FMO), arbitrary slice ordering (ASO), and redundant slices (RS)), and context adaptive variable length coding (CAVLC). It does not contain bidirectional (B), switching predictive (SP), and switching intra (SI) slices, interlace coding tools or context-adaptive binary arithmetic coding (CABAC) entropy coding. The extended profile is a super-set of baseline, adding B, SP and SI-slices and interlace coding tools to the set of baseline profile coding tools and adding further error resilience support in the form of data partitioning (DP). It does not include CABAC. The main profile includes I, P and B-slices, interlace coding tools, CAVLC and CABAC. It does not include enhanced error resilience tools (FMO, ASO, RS, and DP) or SP and SI-slices.

Table 2.1: H.264/AVC profiles.

Coding Tools	Baseline	Main	Extended
I and P Slices	✓	✓	✓
CAVLC	✓	✓	✓
CABAC		✓	
B Slices		✓	✓
Interlaced Coding (PicAFF, MBAFF)		✓	✓
Enhanced Error Resil. (FMO, ASO, RS)	✓		✓
Further Enh. Error Resil (DP)			✓
SP and SI Slices			✓

2.3.7 MPEG-7

MPEG-7 [26] is a different kind of standard as it is a multimedia content description standard, and does not deal with the actual encoding of moving pictures and audio. With MPEG-7, the content of the video (or any other multimedia) is described and associated with the content itself, for example to allow fast and efficient searching in the material. MPEG-7 uses extensible markup language (XML) to store metadata, and it can be attached to a timecode in order to tag particular events in a stream. Although MPEG-7 is independent of the actual encoding technique of the multimedia, the representation that is defined within MPEG-4, i.e. the representation of audio visual data in terms of objects, is very well suited to the MPEG-7 standard. MPEG-7 is relevant for video surveillance since it could be used for example to tag the contents and events of video streams for more intelligent processing in video management software or video analytics applications.

2.3.8 MPEG-21

MPEG-21 [27] is a standard that defines means of sharing digital rights, permissions, and restrictions for digital content. It aims at defining an open framework for multimedia applications. MPEG-21 is ratified in the standards ISO/IEC 21000 - Multimedia framework (MPEG-21). MPEG-21 is a XML-based standard, and is developed to counter illegitimate distribution of digital content.

MPEG-21 is based on two essential concepts: the definition of a fundamental unit of distribution and transaction, which is the digital item, and the concept of users interacting

with them. Digital items can be considered the kernel of the multimedia framework and the users can be considered as who interacts with them inside the multimedia framework. At its most basic level, MPEG-21 provides a framework in which one user interacts with another one, and the object of that interaction is a digital item. Due to that, we could say that the main objective of the MPEG-21 is to define the technology needed to support users to exchange, access, consume, trade or manipulate digital items in an efficient and transparent way.

2.4 Frame Types

The basic principle for video compression is the image-to-image prediction. The first image is called an I-frame and is self-contained, having no dependency outside of that image. The following frames may use part of the first image as a reference. An image that is predicted from one reference image is called a P-frame and an image that is bidirectionally predicted from two reference images is called a B-frame.

1. I-frames: Intra predicted, self-contained;
2. P-frames: Predicted from last I or P reference frame;
3. B-frames: Bidirectional; predicted from two references one in the past and one in the future, and thus out of order decoding is needed;

Figure 2.2 shows how a typical sequence with I-, B-, and P-frames may look. Note that a P-frame may only reference a preceding I- or P-frame, while a B-frame may reference both preceding and succeeding I- and P-frames. The video decoder restores the video by decoding the bit stream frame by frame. Decoding must always start with an I-frame, which can be decoded independently, while P- and B-frames must be decoded together with current reference image(s).

2.5 Group of Pictures

One parameter that can be adjusted in MPEG-4 is the Group of Pictures (GOP) length and structure, also referred to as Group of Video (GOV) in some MPEG standards. It is normally repeated in a fixed pattern, for example:

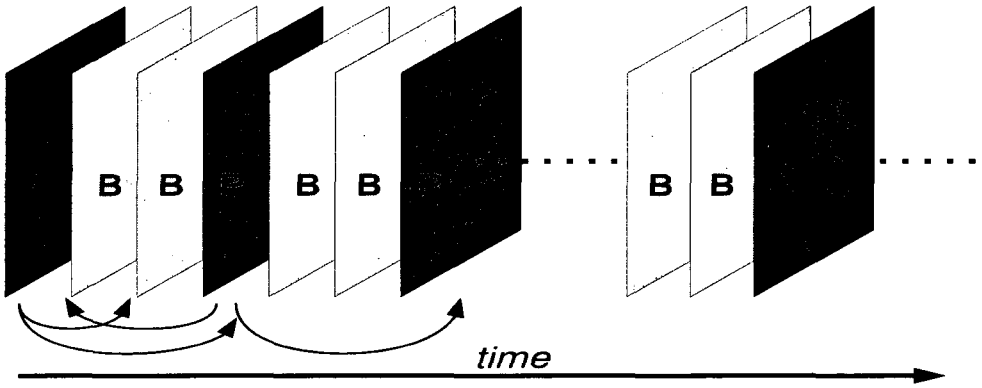


Figure 2.2: Picture types.

1. GOV = 4, e.g. IPPP IPPP ... ;

2. GOV = 15, e.g. IPPPPPPPPPPPPPP IPPPPPPPPPPPPPP ... ;

3. GOV = 8, e.g. IBPBPBPB IBPBPBPB ... ;

The appropriate GOP depends on the application. By decreasing the frequency of I-frames, the bit rate can be reduced. By removing the B-frames, latency can be reduced. The number of frames between two I-frames, can be adjusted to fit the application.

2.6 Variable and Constant Bit Rate

Another important aspect of MPEG is the bit rate mode that is used. In most MPEG systems, it is possible to select the mode, constant bit rate (CBR) or variable bit rate (VBR). The optimal selection depends on the application and available network infrastructure. With limited bandwidth available, the preferred mode is normally CBR as this mode generates a constant and predefined bit rate. The disadvantage with CBR is that image quality will vary. While the quality will remain relatively high when there is no motion in a scene, it will significantly decrease with increased motion. With VBR, a predefined level of image quality can be maintained regardless of motion or the lack of it in a scene. This is often desirable in video surveillance applications where there is a need for high quality, particularly if there is motion in a scene. Since the bit rate in VBR may vary even when an average target bit rate is defined the network infrastructure (i.e. available bandwidth) for such a system needs to have a higher

capacity.

2.7 MPEG Standards Comparison

Looking at MPEG-2 and later standards, it is important to bear in mind that they are not backwards compatible, i.e. strict MPEG-2 decoders/encoders will not work with MPEG-1. Neither will H.264 encoders/decoders work with MPEG-2 or previous versions of MPEG-4, unless specifically designed to handle multiple formats. However, there are various solutions available where streams encoded with newer standards can sometimes be packetized inside older standardization formats to work with older distribution systems. Since both MPEG-2 and MPEG-4 cover a wide range of picture sizes, picture rates and bandwidth usage, the MPEG-2 introduces a concept called Profile@Level [21]. This is created to make it possible to communicate compatibilities among applications. For example, the studio profile of MPEG-4 is not suitable for a PDA and vice versa. Note that: MPEG-2, MPEG-4 and H.264/AVC are all subject to licensing fees.

Since the H.261/H.263 recommendations are neither international standards nor offer any compression enhancements compared to MPEG standards, they are not of any real interest and are not recommended as suitable techniques for video surveillance. MPEG-1 is considered, in most cases, more effective than Motion JPEG. However, for just a slightly higher cost, MPEG-2 provides even more advantages and supports better image quality, comprising of frame rate and resolution. On the other hand, MPEG-2 requires more network bandwidth consumption and is a technique of greater complexity. MPEG-4 is developed to offer a compression technique for applications demanding less image quality and bandwidth. It is also able to deliver video compression similar to MPEG-1 and MPEG-2, i.e. higher image quality at higher bandwidth consumption.

If the available network bandwidth is limited, or if video is to be recorded at a high frame rate and there are storage space restraints, MPEG may be the preferred option rather than motion JPEG. It provides a relatively high image quality at a lower bit rate (bandwidth usage). Still, the lower bandwidth demands come at the cost of higher complexity in encoding and decoding, which in turn contributes to a higher latency when compared to motion JPEG.

Looking ahead, it is not a bold prediction that H.264 will be a key technique for compression

of motion pictures in many application areas, including video surveillance. As mentioned above, it has already been implemented in as diverse areas as high-definition DVD (HD-DVD and Blu-ray), for digital video broadcasting including high-definition TV (HDTV), in the 3rd generation partnership project (3GPP) standard for the third generation mobile telephony and in software such as QuickTime and Apple Computer's MacOS X operating system.

H.264 is now a widely adopted standard, and represents the first time that the ITU, ISO and IEC have come together on a common, international standard for video compression [21]. H.264 entails significant improvements in coding efficiency, latency, complexity and robustness. It provides new possibilities for creating better video encoders and decoders that provide higher quality video streams at maintained bit rate (compared to previous standards), or, conversely, the same quality video at a lower bit rate.

There will always be a market need for better image quality, higher frame rates, and higher resolutions with minimized bandwidth consumption. H.264 offers this, and as the H.264 format becomes more broadly available in network cameras, video encoders and video management software, system designers and integrators will need to make sure that the products and vendors they choose support this new open standard. And for the time being, network video products that support several compression formats are ideal for maximum flexibility and integration possibilities.

2.8 Performance Metric

Distortion measures the difference between the decoded image and the original image. Peak signal to noise ratio (PSNR) is normally used to evaluated the distortion. PSNR is defined by the following formula.

$$E_{MSE} = \frac{1}{m \times n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |I(i,j) - \hat{I}(i,j)|^2 \tag{2.1}$$

$$PSNR = 20 \times log_{10}(\frac{255}{\sqrt{E_{MSE}}}) \tag{2.2}$$

Where, E_{MSE} is the mean square error between an original $m \times n$ image I and its decoded image $\hat{I}$.

2.9 Conclusions

In this chapter, the video coding fundamentals have been presented. It starts with video coding standards. Some basic terminologies and performance metric are also defined in this chapter. These concepts help to understand the following chapters.

Chapter 3

Fast Implementation of H.264 4x4 Intra Prediction

3.1 Introduction

The Joint Video Team (JVT) of ISO/IEC MPEG and ITU-VCEG jointly developed a new video compression standard H.264/AVC [21]. Compared to previous standards, such as MPEG-2, H.263 and MPEG-4, aggressive compression techniques are employed in H.264/AVC standard. As a result, its performance is greatly improved in terms of the compression efficiency, however, this is achieved at the expense of increasing the computational complexity.

Intra prediction utilizes the spatial correlation in an image to predict the block being encoded from its nearby pixels. It is recognized to be one of the main factors contributing the success of H.264/AVC [21]. To select the best prediction mode, the encoder has to search all possible prediction modes exhaustively in order to encode blocks. As a result, the computational complexity in H.264/AVC is extremely high. Some previous approaches to reduce computation complexity of H.264/AVC intra prediction by optimizing and speeding up are presented in [28–36].

In this chapter, a novel parallel architecture to achieve fast intra prediction is presented. The remaining sections are organized as follows: In Section 3.2, the H.264/AVC intra prediction is briefly introduced. Section 3.3 presents the proposed architecture, a redundancy reduction methodology and a simplified mode decision criteria is presented to low complexity computation

of intra frame prediction. Section 3.4 provides simulation results and conclusions are addressed in Section 3.5.

3.2 H.264/AVC Intra Prediction

In H.264/AVC baseline intra coding, two intra prediction modes for luminance component are supported in each profile [21]. One is intra4x4 mode and the other is intra16x16 mode. The intra8x8 is a new prediction type defined in H.264/AVC FREXt [37]. For intra4x4 mode, each MB is divided into sixteen non-overlapping 4x4 blocks. Each block can select one of nine prediction modes. For intra16x16 mode, four prediction modes are available for each MB. Chroma intra prediction is independent to luminance prediction mode. Two chroma components are simultaneously predicted with the same mode. The intra4x4 is more accurate than intra16x16, however it requires more bits to be coded. Hence, intra4x4 is used for highly textured regions while intra16x16 is used for plain regions.

From the complexity perspective, H.264/AVC encodes macroblocks (MBs) by iterating all the luminance intra decisions for each possible chroma intra prediction mode to achieve the best coding efficiency. Therefore, the number of mode combination for luminance and chroma components in a MB is C8 $\times$ (L4 $\times$ 16 + L16), where C8, L4, and L16 represent the number of modes for chroma prediction, 4x4 luminance prediction, and 16x16 luminance prediction respectively. This means that, $4 \times (9 \times 16 + 4) = 592$ different rate distortion optimization (RDO) costs have to be calculated before a best mode can be determined. If the 8x8 luminance prediction in H.264/AVC FREXt is included, the number of mode combination is C8 $\times$ (L4 $\times$ 16 + L8 $\times$ 4 + L16) $= 4 \times (9 \times 16 + 9 \times 4 + 4) = 736$. Even in low complexity mode [38], $9 \times 16 + 4 + 4 \times 2 = 156$ mode decisions are needed. Although low complexity mode can greatly reduce computational load, encoding time still needs to be reduced for some applications requiring very low delay. If the target application is high definition TV (HDTV), each frame needs $(1920 \times 1080 \times 592) / 256 = 4{,}795{,}200$ calculations, which is not feasible for real-time implementation. Thus, speeding up intra coding process is essentially required.

In H.264/AVC [21], intra prediction is performed in two modes, intra4x4 and intra16x16 mode. The luminance samples in a MB is divided into sixteen 4x4 blocks. Then these blocks are coded sequentially as showed in Fig. 3.1.

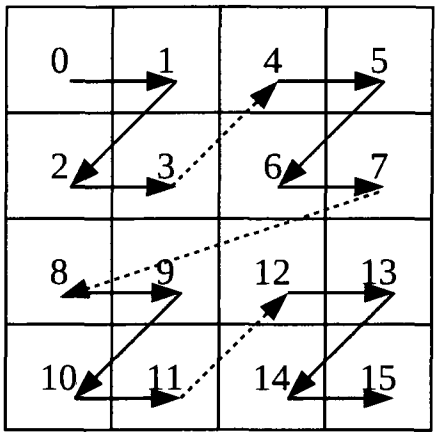


Figure 3.1: Intra4x4 prediction order.

There are nine prediction modes for each 4x4 block in intra4x4 prediction mode as shown in Fig. 3.2. They are one DC mode (mode2) and eight directional modes, e.g. vertical (mode0), horizontal (mode1), diagonal down left (mode3), diagonal down right (mode4), vertical right (mode5), horizontal down (mode6), vertical left (mode7) and horizontal up (mode8). To reflect the edge trend of the block, the prediction for the current 4x4 block is calculated using the boundary pixels of the previously decoded blocks above and to the left of it. Since the pixels along the direction of the local edge have similar values, an accurate prediction can be achieved if the direction of the prediction mode is the same as the edge direction of the block. In this figure, neighboring samples used for prediction are labeled with capital letters A M. The 16 grey grids are the predicted samples called predictors. Each predictor is extrapolated from the neighboring pixels A M. The extrapolation process is specified by H.264/AVC [21]. Definitions of extrapolation equations for nine prediction modes are listed in Table B.1. Because each 4x4 block use neighboring samples to form predictors, the encoder needs to reconstruct the current block before moves to the next block. Therefore, in the intra4x4 prediction, the block in the upper left corner is processed at first and the lower right corner is processed at last. The intra prediction for each block uses the pixels in its left and top sides as reference pixels. A block thus can not be predicted until its previous block has been reconstructed. The reconstruction includes discrete cosine transform (DCT), quantization (Q), inverse quantization (IQ), and inverse discrete cosine transform (IDCT).

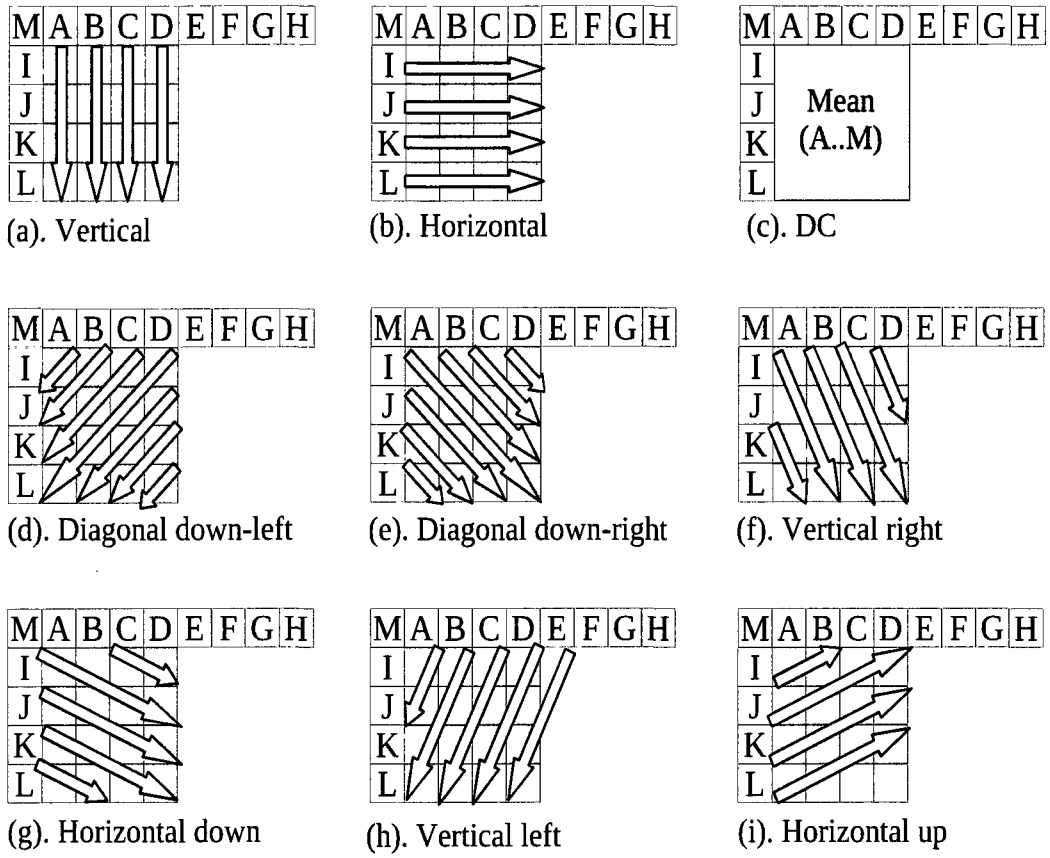


Figure 3.2: Intra4x4 prediction modes.

3.3 Proposed Parallel Architecture & Methodology

The original process handles blocks in serial, which is not efficient as illustrated in Fig. 3.3 (a). Efficient architectures have been reported in [28–31], however, they all have drawbacks either with the pipelining architecture or in compression gains. Huang's work [28] has bubbles between Intra4x4 predictions because of the low throughput of reconstruction process so that the prediction has to wait for the completion of reconstruction. Lee's work [29] perfectly pipelines the intra prediction and reconstruction process shown as Fig. 3.3 (b), however, it requires that both intra prediction and reconstruction have exact equal processing cycles. It also reduces some prediction modes in some blocks in order to enforce pipelining, hence, the video quality is degraded. Jin's work [30] proposes both partially and fully pipelined architectures for intra4x4 prediction and has the same drawback as the approach in [29]. Moreover, the architectures add dependency graph process in order to improve gains, however,

this increases hardware overhead. It takes 25 cycles to process each block, which is too long for high throughput reconstruction. Suh's work [31] is similar to Huang's work, which takes 34 cycles to process each block. The thesis proposes an efficient parallel architecture followed by a redundancy reduction algorithm to speed up the intra4x4 prediction.

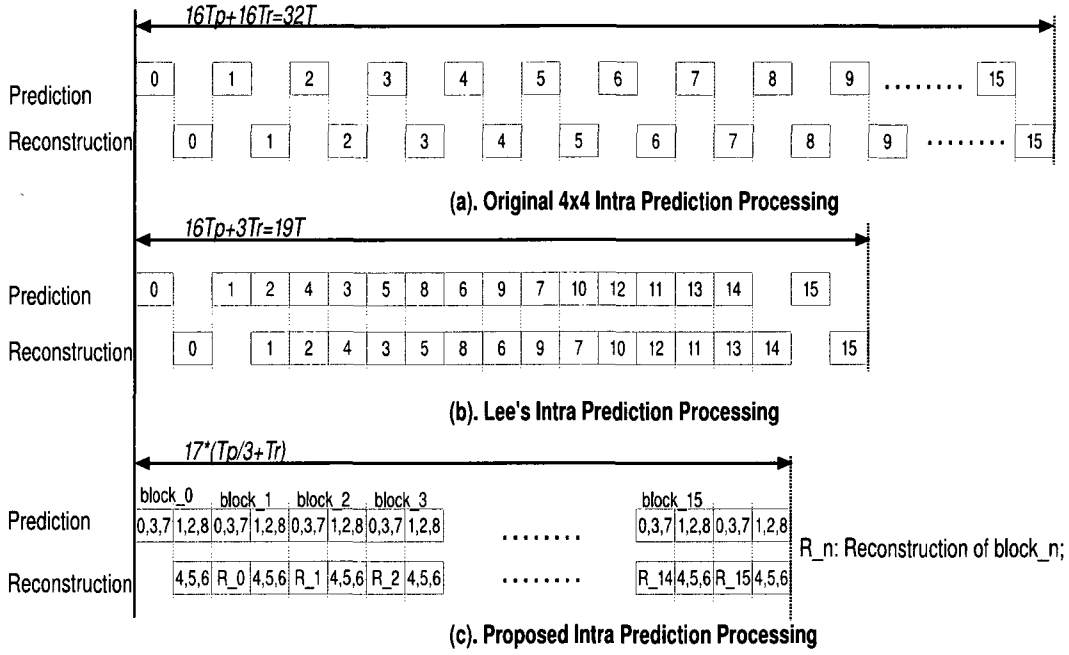


Figure 3.3: Intra4x4 prediction process.

3.3.1 Parallel Architecture

Although a data dependency truly exists among blocks in intra4x4 prediction, we can state, after careful observation, that such data dependency does not exist in some intra4x4 prediction modes. Therefore, they are able to be processed without waiting for their previous blocks to be reconstructed, i.e., mode0, 3, and 7 of the current block can be simultaneously predicted when its previous block is being reconstructed. After the reconstruction of its previous block is complete, the rest of modes, i.e., model, 2, 8 and mode4, 5, 6 of the current block, shown as Fig. 3.3 (c) can be predicted in parallel. The same procedure follows in the rest of the blocks. To sum up, we divide nine prediction modes into three groups, and each group has three prediction modes.

The proposed architecture has four advantages compared to previous works. The first advantage is that the proposed process does not ignore any prediction modes. The second advantage is that the encoder follows that order specified in H.264/AVC standard to guarantee the consistency between the encoder and the decoder. The third advantage is that it does not require the processing cycles of intra prediction and reconstruction to be exact the same. The last advantage is that the proposed architecture can reduce total processing time of each MB to $17 \times (1/3T_p + T_r)$ if high throughput reconstruction is adopted, where T_p is prediction time and T_r is reconstruction time.

As shown in Fig. 3.4, the luma4x4 prediction unit mainly consists of five functional blocks for Prediction Generator-1, Prediction Generator-2, SAD (sum of absolute difference) Computation, Reconstruction, and Controller. Prediction Generator-1 and Prediction Generator-2 calculate the predicted pixel values for all the intra modes. SAD Computation block calculates SAD values for each mode in order to make mode decision. Reconstruction block recovers the prediction pixels of the best mode by the reconstruction process (DCT, Q, IQ, and IDCT). The Controller block selects the right pixels from Edge Pixels buffer and feeds them into Prediction Generator blocks.

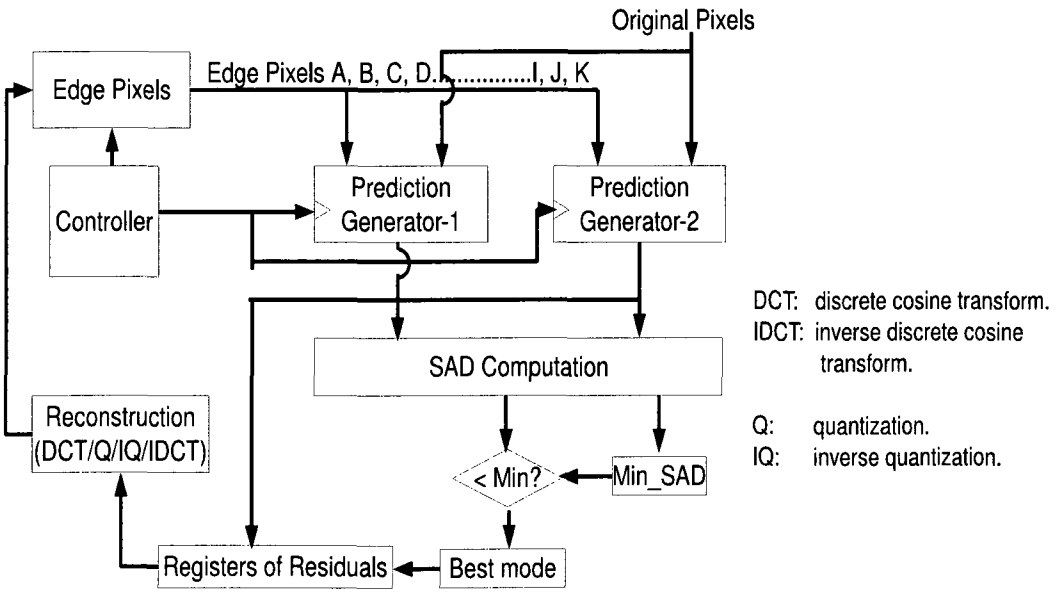


Figure 3.4: Intra4x4 prediction architecture.

In the first step, the Generator-1 is parallel with the reconstruction process and the first group (mode0, 3 and 7) are predicted in this step. In the second step, the Generator-1 and Generator-2 are parallel to process the second group (mode1, 2, and 8) and the third group (mode4, 5 and 6). The output of Edge Pixels unit are selected by the controller. Each group has its own best mode by calculating SAD. The final best mode is obtained by comparing the best mode of each group. Meanwhile, the residuals of this block with the best mode are written into the register. The reconstruction process implements DCT, Q, IQ and IDCT based on the residuals. After added to the values of prediction, the reconstructed edges pixels are stored in the buffer for being used for predicting the next block.

3.3.2 Redundancy Reduction Algorithm

Considering the definition of the nine intra4x4 prediction modes of H.264/AVC [21] as shown in Appendix Table B.1, there are some identical parts in calculating the predicted values. It is possible to reduce memory access and improve prediction time by eliminating these redundancy computations. A detailed summary is listed in Table 3.1. In this table, identical prediction items exist not only in different pixels of the same mode, but also in different pixels of different modes, for example, the predicted value of both pixels (1, 0) and (0, 1) in diagonal down left prediction mode is equal to the value in pixel (3, 0) of diagonal down right prediction mode. Therefore, the redundancy computations are able to be reduced.

Table 3.1: Reducing intra4x4 prediction redundancy.

Mode	Equation	Round	Shift	Positions (x, y)
Vertical	A	0	0	(0,0) (1,0) (2,0) (3,0)
	B	0	0	(0,1) (1,1) (2,1) (3,1)
	C	0	0	(0,2) (1,2) (2,2) (3,2)
	D	0	0	(0,3) (1,3) (2,3) (3,3)
Horizontal	I	0	0	(0,0) (0,1) (0,2) (0,3)
	J	0	0	(1,0) (1,1) (1,2) (1,3)
	K	0	0	(2,0) (2,1) (2,2) (2,3)
	L	0	0	(3,0) (3,1) (3,2) (3,3)
DC	(I+J+K+L+A+B+C+D+4)>>3	4	3	ALL
DDL[1]	(A+2B+C+2)>>2	2	2	(0.0)
	(B+2C+D+2)>>2	2	2	(0,1) (1,0)

	(C+2D+E+2)>>2	2	2	(0,2) (1,1) (2,0)
	(D+2E+F+2)>>2	2	2	(0,3) (1,2) (2,1) (3,0)
	(E+2F+G+2)>>2	2	2	(1,3) (2,2) (3,1)
	(F+2G+H+2)>>2	2	2	(2,3) (3,2)
	(G+3H+2)>>2	2	2	(3,3)
DDR^2	(L+2K+J+2)>>2	2	2	(3,0)
	(K+2J+I+2)>>2	2	2	(2,0) (3,1)
	(J+2I+M+2)>>2	2	2	(1,0) (2,1) (3,2)
	(I+2M+A+2)>>2	2	2	(0,0) (1,1) (2,2) (3,3)
	(M+2A+B+2)>>2	2	2	(0,1) (1,2) (2,3)
	(A+2B+C+2)>>2	2	2	(1,3)
	(B+2C+D+2)>>2	2	2	(0,3)
VR^3	(M+A+1)>>1	1	1	(0,0) (2,1)
	(A+B+1)>>1	1	1	(0,1) (2,2)
	(B+C+1)>>1	1	1	(0,2) (2,3)
	(C+D+1)>>1	1	1	(0,3)
	(K+2J+I+2)>>2	2	2	(3,0)
	(J+2I+M+2)>>2	2	2	(2,0)
	(I+2M+A+2)>>2	2	2	(1,0) (3,1)
	(M+2A+B+2)>>2	2	2	(1,1) (3,2)
	(A+2B+C+2)>>2	2	2	(1,2) (3,3)
	(B+2C+D+2)>>2	2	2	(1,3)
HD^4	(L+K+1)>>1	1	1	(3,0)
	(K+J+1)>>1	1	1	(2,0) (3,2)
	(J+I+1)>>1	1	1	(1,0) (2,2)
	(I+M+1)>>1	1	1	(0,0) (1,2)
	(L+2K+J+2)>>2	2	2	(3,1)
	(K+2J+I+2)>>2	2	2	(2,1) (3,3)
	(J+2I+M+2)>>2	2	2	(1,1) (2,3)
	(I+2M+A+2)>>2	2	2	(0,1) (1,3)
	(M+2A+B+2)>>2	2	2	(0,2)
	(A+2B+C+2)>>2	2	2	(0,3)
VL^5	(A+B+1)>>1	1	1	(0,0)
	(B+C+1)>>1	1	1	(0,1) (2,0)
	(C+D+1)>>1	1	1	(0,2) (2,1)
	(D+E+1)>>1	1	1	(0,3) (2,2)

	(E+F+1)>>1	1	1	(2,3)
	(A+2B+C+2)>>2	2	2	(1,0)
	(B+2C+D+2)>>2	2	2	(1,1) (3,0)
	(C+2D+E+2)>>2	2	2	(1,2) (3,1)
	(E+2F+G+2)>>2	2	2	(1,3) (3,2)
	(F+2G+H+2)>>2	2	2	(3,3)
HU[6]	L	0	0	(2,2) (2,3) (3,0) (3,1) (3,2) (3,3)
	(L+K+1)>>1	1	1	(1,2) (2,0)
	(K+J+1)>>1	1	1	(0,2) (1,0)
	(J+I+1)>>1	1	1	(0,0)
	(3L+K+2)>>2	2	2	(1,3) (2,1)
	(L+2K+J+2)>>2	2	2	(0,3) (1,1)
	(K+2J+I+2)>>2	2	2	(0,1)

Figure 3.5 illustrates how to calculate 6 prediction modes (except DC, vertical and horizontal prediction modes) with the common parts. The 14 common parts are listed on the left side of Fig. 3.5. The terms ("A" to "M") of the common parts indicate the neighbouring pixels as shown in Fig. 3.2. The numbers N_{xy} on the right side in Fig. 3.5 refer to mode N in position (x, y). For example, the predicted value in pixel (1, 1) of diagonal down left mode, (A+2B+C+2)>>2, can be calculated by adding "(A+B)" and "(B+C)". By analysis, only 14 common parts and 23 equations of their combination are required for intra4x4 prediction calculations, which can be implemented by 27 adders and 23 shifts. Moreover, the DC prediction mode is very straightforward, which only requires 3 adders and 1 shift. For vertical, horizontal and part of horizontal up prediction modes, the predicted values can be obtained only by propagating the values of edge pixels. Therefore, for total intra4x4 mode prediction, the proposed algorithm requires 30 adders and 24 shifts. It can be completed within one cycle.

To achieve fast intra4x4 prediction, a high throughput reconstruction process is also required. The reconstruction process (DCT, Q, IQ, and IDCT) is implemented in parallel with

[1]Diagonal down-left
[2]Diagonal down-right
[3]Vertical right
[4]Horizontal down
[5]Vertical left
[6]Horizontal up

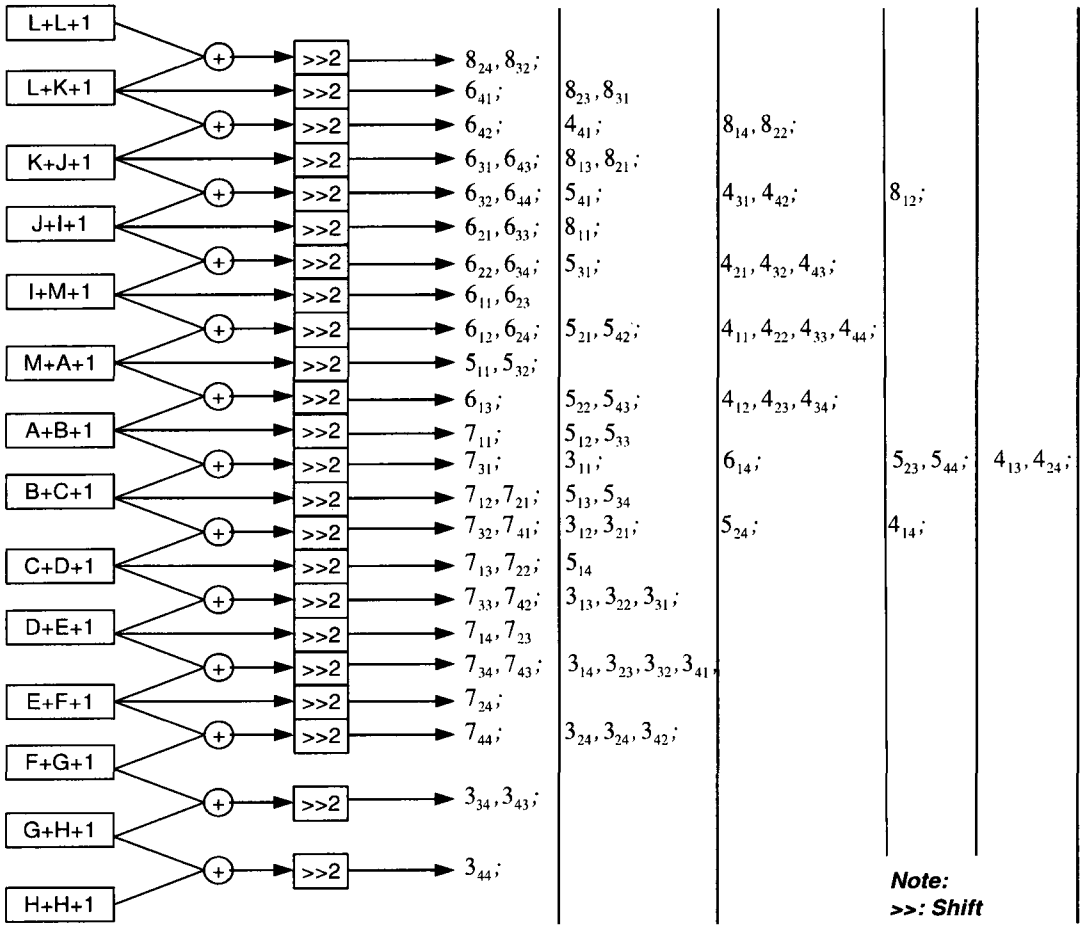


Figure 3.5: Redundancy reduction algorithm

three intra4x4 prediction modes (mode0, 3, and 7). Our previous work [39] of a high throughput realization of DCT and IDCT is adopted in this implementation. It takes only one cycle to process DCT and IDCT, separately. Fast quantization and its inverse have been implemented using the approach in [40], in which a look-up table is utilized to complete quantization in one cycle. In the proposed design, both fast DCT/IDCT and IQ are required in order to achieve fast implementation. The total reconstruction time to process a MB is 6 cycles (2 cycles for control).

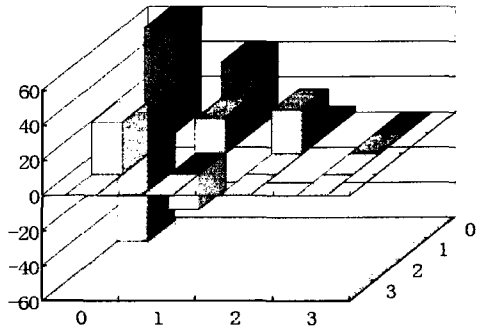
3.3.3 Complexity Reduced Mode Decision Algorithm

In H.264/AVC intra prediction, both SAD and sum absolute transform difference (SATD) [41] are widely accepted for mode decision. However, they have high computational complexity

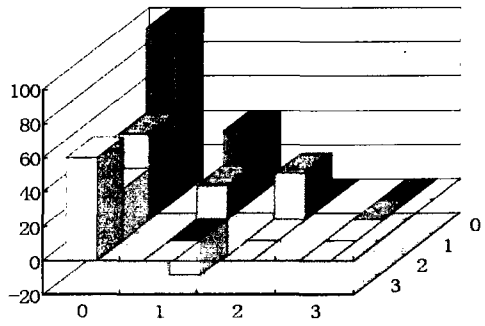
and are not adoptable for fast computation, particularly not for hardware implementation. To speed up the intra4x4 prediction process, a low complexity mode decision criteria is required. After carefully observation, we find that their quantized residual values are often centered at zero when the blocks have lower distortion. Based on the observation, a new cost function, sum of difference and symmetry (SDS), can be given by (3.1), where $R_{max} - R_{min}$ represents the difference of each mode of this block while $R_{max} + R_{min}$ represents the symmetry of the mode. They are given different weights depends on the importance.

$$Cost_{SDS} = |R_{max} - R_{min}| << 1 + |R_{max} + R_{min}| \tag{3.1}$$

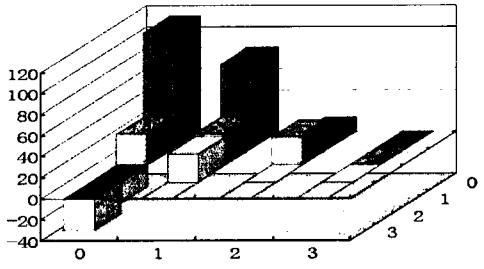
where, R_{max} and R_{min} are the maximum and minimum residuals, respectively. An experiment is implemented to compare the costs between "SDS" and "SATD" as shown in Fig. 3.6. The result shows that mode 0 has the minimal SATD value and the minimal "SDS" value at the same time. This can be explained by observing that the residues of model are distributed quite close to "0" with the symmetrical property. Some experiments have been done to compared the best mode decided by computing SATD with computing SDS. The experimental results show that the probability of SDS is slightly lower than SATD on average, which means that the behavior of SDS is more like that of SATD, and it is able to be a new criteria to decide the best mode.



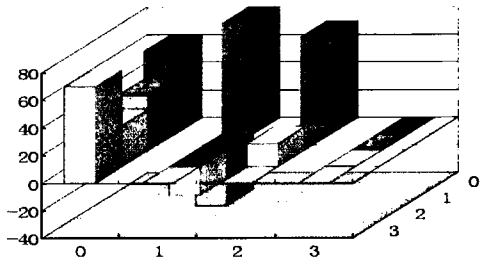
(a). Mode0. SATD=868, SDS=108.



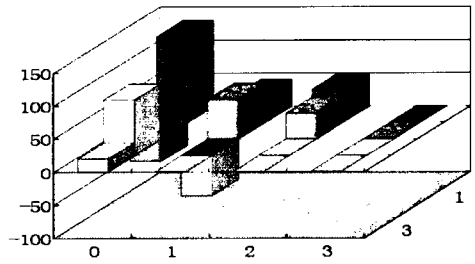
(b). Model. SATD=1178, SDS=287.



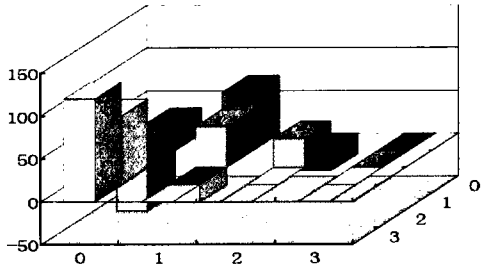
(c). Mode2. SATD=1102, SDS=268.



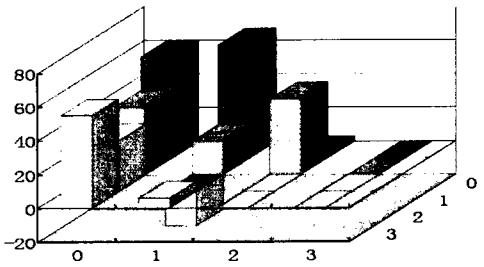
(d). Mode3. SATD=1436, SDS=314.



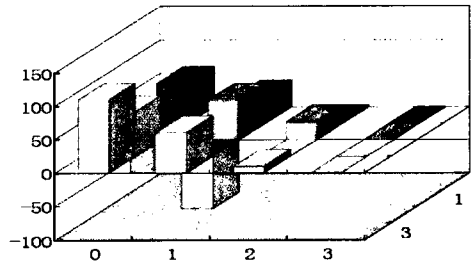
(e). Mode4. SATD=1268, SDS=329.



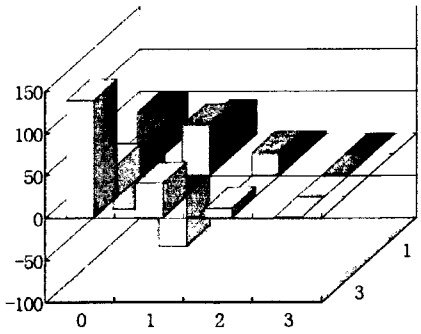
(f). Mode5. SATD=1130, SDS=286.



(g). Mode6. SATD=1134, SDS=306.



(h). Mode7. SATD=1173, SDS=306.



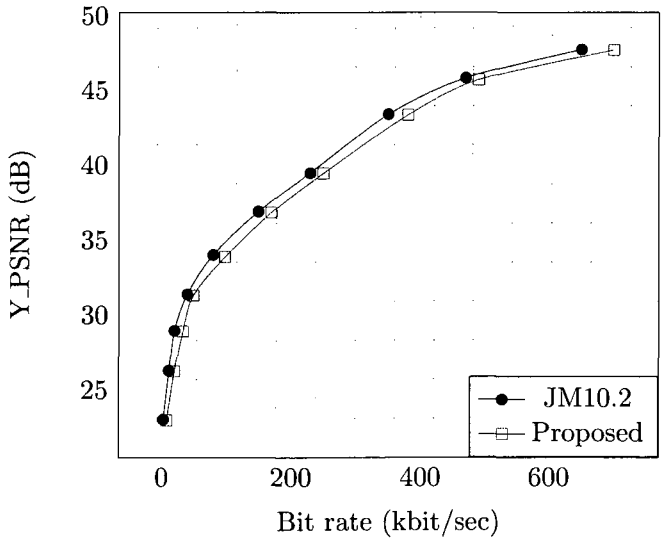
(i). Mode8. SATD=1206, SDS=307.

Figure 3.6: Comparison of SDS and SATD costs

3.4 Experimental Results and Analysis

The simulation results reported in this subsection are aimed at evaluating the performance of the SDS algorithm.We use JM10.2 [38] reference software to evaluate the compression efficiency and execution speed of our proposed parallel algorithm. The experiments conditions are: baseline profile, QCIF frame size, 30 frames/second, intra frame only, RDO off, frame number equal to 100. Several sequences have been verified including "Foreman", "Akiyo", "Coastguard", "News and Calendar", "Football", and "Table tennis". The experimental results of sequences "News" and "Akiyo" are shown in Fig. 3.7, and no significant performance degradation has been found with our proposed criteria. These experimental results of the rest of sequences are attached in Appendix Fig. B.1 and Appendix Fig. B.2. It is clear from these plots that the proposed SDS algorithm achieves comparable rate-distortion performance to the reference with SAD algorithm. The performance loss relative to the reference in terms of PSNR is less than 0.5 dB for "Football" and "Table tennis", about 0.3 dB for "Coastguard" and "Foreman", and about 0.15 dB for "Akiyo" and "News and Calendar". Compared to the reference code with SAD, the complexity saving to compute SDS is more than 85%. The complexity is measured using consumed CPU time, and the computational saving is similar for all sequences. Moreover, the proposed "SDS" criteria algorithm is able to be implemented by hardware easily since it only requires addition and shift operations.

(a). News QCIF Performance Comparison



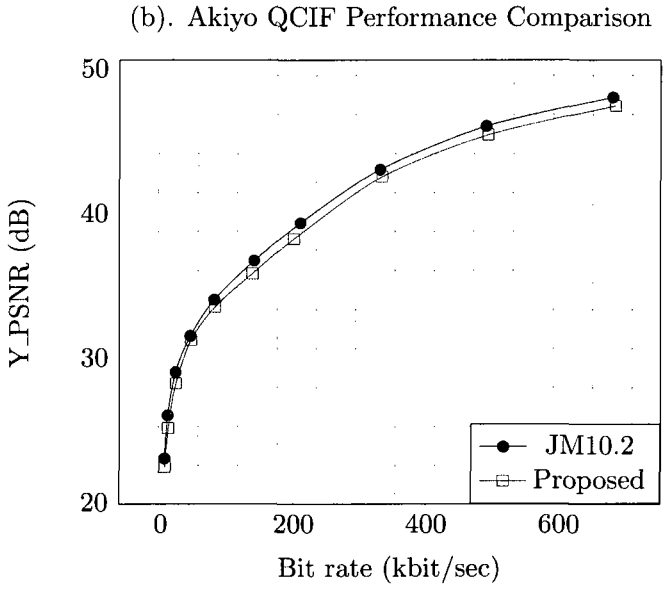


Figure 3.7: News & Akiyo performance comparison

Table 3.2 shows the experimental results compared to the previous works. The proposed architecture reduces complexity up to 79% compared to [28], 63% compared to [31] and 57% compared to [30]. By comparing average peak signal-to-noise ratio (PSNR) of the proposed approach with H.264/AVC reference module at various bit rates of sequences, we find no significant quality degradation. The proposed parallel architecture has been implemented in a Xilinx Virtex-4 FPGA using Xilinx ISE Series 9.1i. The implementation is verified with register transfer level (RTL) simulations using Mentor Graphics ModelSim SE 6.1.

Table 3.2: Execution cycles for each MB

Methods	Cycles/Block	Cycles/MB	Full Modes	Savings
Huang. [28]	60	960	Yes	79%
Suh. [31]	34	544	Yes	63%
Jin. [30]	25	475	No	57%
Proposed	12	204	Yes	- -

3.5 Conclusions

A fast parallel architecture, redundancy reduction algorithm and a new criteria for mode decision of H.264/AVC intra4x4 prediction have been proposed in this chapter. This parallel execution cuts down part of data dependency. It has adopted high-throughput DCT/IDCT and

Q/IQ to approach a fast implementation resulting in reducing the intra prediction execution time up to 79% compared with the previous works. Meanwhile, no any prediction modes are ignored. Software simulation shows no significant performance degradation. In order to verify the proposed design, it has been implemented in Xilinx Virtex-4 FPGA.

Chapter 4

H.264/AVC Rate Control Algorithms

4.1 Introduction

Block-based hybrid video encoding schemes such as the MPEG [16, 18, 20] and H.26* [21, 42] families are inherently lossy compression processes. They achieve compression not only by removing truly redundant information from the bitstream, but also by making small quality compromises in ways that are intended to be minimally perceptible. In particular, the quantization parameter (QP) regulates how much spatial detail is saved. When QP is very small, almost all that detail is retained. As QP is increased, some of that detail is aggregated so that the bit rate drops – but at the price of some increase in distortion and some loss of quality. Figure 4.1 (a) suggests that the relationship for a particular input picture – if you want to lower bit rate, you can do so by increasing QP at a cost of increased distortion. Figure 4.1 (b) suggests that as source complexity varies during a sequence, you move from one curve to the another.

In H.264/AVC standard [21], the output bit rate and video quality of a video encoder depend on several coding parameters such as QP value and coding mode [43]. The QP value scaling transform coefficients is used to regulate the bit rate to meet the requirements of limited bandwidth channel while maximizing the video quality. This process is well-known as RC. Unlike other normative techniques in H.264/AVC, such as multiple reference frames, variable

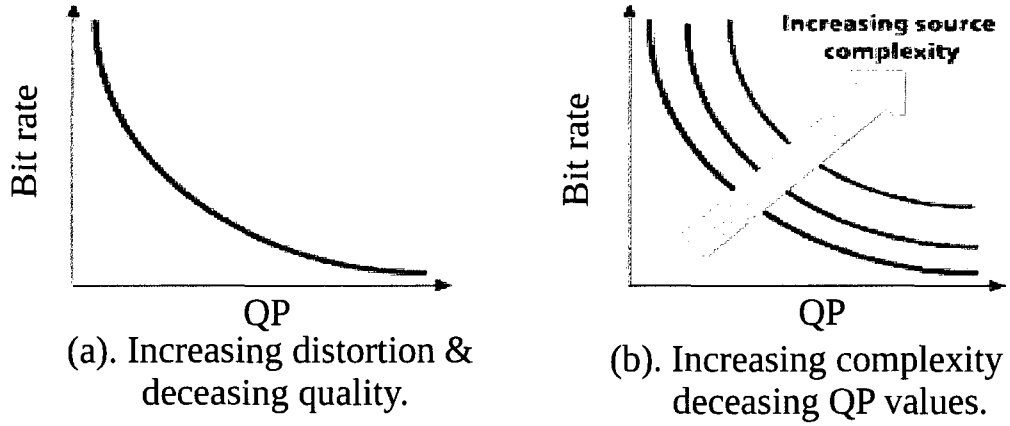


Figure 4.1: Relationship between bit rate and QP

block size, quarter sub-pixel motion compensation and deblock filtering, RC is a no-normative tool and unspecified in H.264/AVC standard because the following dilemma [44]: The QP value should be available before selecting the best mode for each macroblock (MB), which is the one that can minimize the overall Lagrangian cost function $J = D + \lambda \times R$ [21], where λ is the Lagrangian multiplier, D and R are the mean square error and bit rate of each MB. However, the QP value can not be accurately obtained before the MB has been encoded. To break or circumvent the dilemma, estimated QP has to be used in H.264/AVC RC schemes. To estimate QP accurately, a rate-quantization (R-Q) model [45] is normally employed in H.264/AVC RC. Temporal correlation is mostly exploited in the R-Q model. Thus, there are existing issues: First, the initial QP (QP of the first frame) is simply determined by bits per pixel (BPP) as there is no temporal information available at this moment. However, the initial QP has a great impact on the following frames. The worse selection of the initial QP may highly exceed buffer budgets so that the encoder attempts to salvage the bits of the following frames, which leads to an abrupt degradation in video quality of the latter frames or even frame skips. Second, the QP value of intra frame is obtained by taking the average of QP values of its previous global of pictures (GOPs). The average assumes that the intra frame has high correlation with its previous frames and ignores its own complexity of the current frame. Therefore, the determination should be inaccurate. Finally, the output fluctuation occurs during scene changes so that video quality is degraded.

To resolve the above issues, the following QP determination algorithms [45–49] have been

proposed for initial frame, but not for intra frame. Strictly speaking, initial frame is also intra frame and its QP determination algorithm should be similar to intra frame. However, it is separated from the intra frame because it is the start frame with no available temporal information. The JVT-G012 RC algorithm [45] adopted in H.264/AVC reference model determines the initial QP value only based on bits per pixel (BPP). Wang [46] proposes an initial QP determination scheme by based on BPP, entropy information (EI), and intra16 DC mode (IM) as complexity measures of the current intra frame. Jing [47] proposes a QP determination algorithm based on the gradient complexity measure of the current frame. The optimal binary search algorithm [48] reduces the search processing times from fifty-two initial QP values to six values. In [49], the QP value of intra frame is obtained by efficient bit allocation scheme between I-frame and P-frames. The algorithms in [45, 48, 49] neglect the content of the current frame and thus obtain inaccurate QP values while the algorithms in [46, 47] have higher complexity in estimating QP values of intra frame. With respect to all the above drawbacks, this thesis proposes a novel deviation-based QP determination approach to achieve relatively stable constant bit rate (CBR) output and improved performance compared to H.264/AVC reference model JM12.0 [50].

In this chapter, the problems existing in current H.264/AVC rate control schemes are presented at first. Then some relative previous works are reviewed in Section 4.3. The proposed algorithms and methodologies are provided in Section 4.4.3. Conclusions and analysis are presented in the end.

4.2 Existing Problems

To estimate QP accurately, a rate-quantization (R-Q) model is normally employed in H.264/AVC RC. Temporal correlation is mostly exploited in these R-Q models. Thus, there are issues existing: Firstly, the initial QP (QP of the first frame) is simply determined by the bits per pixel (BPP) because no temporal information available. However, the initial QP has a great impact on the following frames. The bad selection of initial QP may highly exceed buffer budgets so that the encoder attempts to salvage the bits of the following frames, which leads to an abrupt degradation in video quality of the latter frames or even frame skips. Secondly, the QP value of intra frame is obtained by the average of QP values of P frames in its previous global

of pictures (GOP). The average assumes that the intra frame has high correlation with its previous frames and ignores its own complexity of the current frame. Thus, the determination should be inaccurate enough. Finally, the issue is even existing in the following inter frames, in which the rate control can not accurately estimate the QP values of the following frames if the complexity of the picture context is ignored, particularly in frames with higher motion or when scene change happens.

4.2.1 The Dilemma of Chicken and Egg

Unlike its previous standards, the QP of H.264/AVC involves both rate control and rate distortion optimization (RDO). There exists a "chicken and egg" dilemma shown in Fig. 4.2 when the rate control is implemented. To perform RDO for a macroblock (MB), a QP should first be determined for the MB by using Mean absolute different (MAD) of the MB and the number of head bits. However, the MAD and head bits are only available after performing RDO. To solve this issue, many previous works have been done to give a more accuracy MAD. Therefore, to perform better rate control in H.264/AVC, MAD, initial QP and head bits have to be estimated accuracy as much as possible. The following section briefly presents the linear adaptive RC method employed in H.264/AVC reference module.

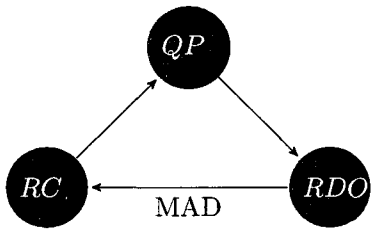


Figure 4.2: Relationship of RC/QP/RDO

4.2.2 PSNR & Output Bit Rate Fluctuation

Rate control mechanism is to map the varying encoder bit rate into the constant bit rate (CBR) channel. Better RC scheme generates video quality and output bit rate with less fluctuation while worse one generates big fluctuation in both video quality and output bit rate. Therefore, accurate QP values are required for a frame to be encoded. However, the

available RC scheme does not take the frame complexity into consideration, which is certainly not accurate enough and must cause average PSNR degradation, output bit rate fluctuation, and even frame skipping, especially for complicated sequences or scene changes and low target bit rates.

4.3 Previous Works

Based on the issues mentioned above, many rate control schemes have been proposed in recent years. Most of these rate control algorithms are focused on inter frames and can be classified into two categories. The first one uses the traditional quadratic rate-distortion (Q2 R-D) model to optimize the allocated bit rates. The second one uses ρ-domain source model to make optimization.

4.3.1 Q2 R-D model

1. At first, we have to mention the most popular scheme, which is employed in JVT-G012 [45]. The one-pass scheme is proposed by Li *et al.*. It utilizes the spatial-temporal correlation to circumvent the "chicken and egg" dilemma, where a linear MAD model is exploited to predict the complexity of the current frame. The target bit rate is thus calculated by the Q2 R-D function 4.2.

$$MAD = \sum_{i,j} |Residuals_{i,j}| = \sum_{i,j} |Source_{i,j} - Prediction_{i,j}| \tag{4.1}$$

$$ResidualBits = \frac{C_1 * MAD}{QP} + \frac{C2 * MAD}{QP} \tag{4.2}$$

   Where, the free coefficients C_1 and C_2 may be estimated empirically.

2. JVT-D030 [51] is a two-pass scheme proposed by Ma *et al.*. In this scheme, a TM-5 [52] alike method is employed in each pass. The first pass is followed by an extra pass process only if the first pass fails to obtain an appropriate QP value.

3. Jiang *et al.*, propose a low delay rate control scheme for H.264/AVC [53]. The scheme

improves the accuracy of MAD by employing PSNR-based complexity estimation. Also, the QP value of each MB is adjusted by the encoded bits. Therefore, the scheme achieves an average 0.66 dB improvement in video quality. The drawback of this scheme is that it is more complicated and not suitable for real-time process.

4.3.2 ρ-domain model

1. Lim's work [54] proposes an ρ-domain (the percentage of zeros among the transformed coefficients) rate control algorithm to estimate the video characteristic and achieve better performance. The algorithm utilizes two kinds of linear regression to estimate the optimum QP value. The more accurate QP value is obtained by $R- > \rho- > E_\rho- > E_{QP}- > QP$ linear regressive model.

2. Tu *et al.* [55] propose other ρ-domain rate control algorithm for H.264/AVC. In the algorithm, the MAD can be approximated by the variance of the $(u, v)th$ transform coefficient $\sigma_Y^2(u, v)$ and the $\sigma_Y^2(u, v)$ can be expressed by ρ.

4.4 Proposed Intra Frame Coding Algorithm

The intra frame QP determination algorithm contains two parts in this thesis: initial QP determination and intra frame QP determination. Strictly speaking, initial frame is also intra frame and its QP determination should be the same as intra frame. However, it is separated from the intra frame because it is the start frame and no temporal information available. In addition, they are processed in different ways in H.264/AVC reference model.

4.4.1 Initial QP Determination Algorithms Review

To resolve the above issues, many algorithms of QP determination [45–49] have been proposed for initial frame, but not for intra frame. Strictly speaking, initial frame is also intra frame and its QP determination algorithm should be similar to intra frame. However, it is separated from the intra frame because it is the start frame and no temporal information available.

1. **H.264/AVC standard intra frame QP determination:**

   In H.264/AVC reference model, the initial QP is determined only based on bit rates per

picture (BPP) as JVT-G012 [45].

2. **Entropy information intra frame QP determination:**

   Wang [46] proposes an initial QP determination scheme by utilizing bit rate per picture (BPP), entropy information (EI) and intra16 DC mode (IM) as complexity measures of the current intra frame.

3. **Gradient-based frame complexity QP determination:**

   Jing [47] proposes a QP determination algorithm based the gradient complexity measure of the current frame.

4. **Binary search of intra frame QP determination:**

   The optimal binary search algorithm [48] reduces the search processing times of the initial QP from 52 indexes to 6 indexes.

5. **Intra frame bit allocation QP determination:**

   In [49], the QP value of intra frame is obtained by efficient bit allocation scheme between I-frame and P-frames.

The above algorithms either neglect the context of the current frame and thus obtain inaccuracy QP values [45, 48, 49] or have higher complexity [46, 47] in estimating QP values of intra frame. With respect to all the above drawbacks, we propose a novel deviation-based QP determination algorithm in this thesis. Meanwhile, two adaptive RC schemes are also presented in order to obtain CBR output. Furthermore, the fluctuation of output bit due to scene change is able to be detected and thus avoided with our algorithm.

4.4.2 MB Deviation Measure

To achieve the best initial QP for each video sequence, all the frames are set to intra frames. We try all the possible initial QP values for the following video sequences under the same BPP (192kb) and find the best QP value for each frame shown in Fig. 4.3. Obviously, the first video frame in different video sequences with the equal BPP value have different best initial

QP values, which means the best initial QP value not only depends on the BPP parameter but also the content of each video sequence.

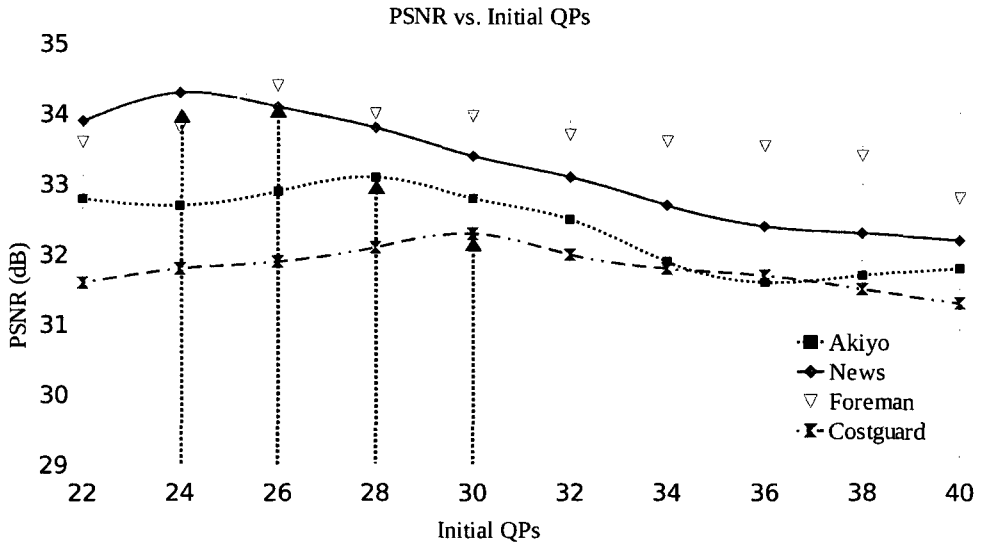


Figure 4.3: Best initial QPs

Our proposed algorithm is based on the observation that the output bit rate of each frame is highly correlated with the deviation-based frame complexity measure. Assuming $Lu(x, y)$ is the value of luminance component of pixel at (x, y), the deviation of a MB (16×16 pixels), which includes four 8×8 blocks, is calculated as:

$$Dev\_MB = \frac{1}{4}\sum_{n=0}^{3} Dev\_Block_n \tag{4.3}$$

$$Dev\_Block_0 = \sum_{y=0}^{7}\sum_{x=0}^{7} |Lu(x,y) - \frac{1}{8\times 8}\sum_{y=0}^{7}\sum_{x=0}^{7} Lu(x,y)| \tag{4.4}$$

$$Dev\_Block_1 = \sum_{y=0}^{7}\sum_{x=8}^{15} |Lu(x,y) - \frac{1}{8\times 8}\sum_{y=0}^{7}\sum_{x=8}^{15} Lu(x,y)| \tag{4.5}$$

$$Dev\_Block_2 = \sum_{y=8}^{15}\sum_{x=0}^{7} |Lu(x,y) - \frac{1}{8\times 8}\sum_{y=8}^{15}\sum_{x=0}^{7} Lu(x,y)| \tag{4.6}$$

$$Dev\_Block_3 = \sum_{y=8}^{15}\sum_{x=8}^{15} |Lu(x,y) - \frac{1}{8\times 8}\sum_{y=8}^{15}\sum_{x=8}^{15} Lu(x,y)| \tag{4.7}$$

where Dev_MB and Dev_Block_n are the deviation values of MB and $block_n$, respectively, where $n = 0$, 1, 2, and 3. To further illustrate the relationship between the deviation and the

output bit rate, we show a set of scatter plots of bit rate versus average deviation value of each MB in Fig. 4.4. Obviously, there exists a linear relationship between these two factors. For the same QP, the larger the deviation value, the more bits are allocated for this frame. Moreover, the slope of each line decreases with increasing the QP value. From this linear correlation between the number of bits and the deviation, we can assume that for a fixed QP, the output bit rate of one intra-coded frame is proportional to its average MB deviation.

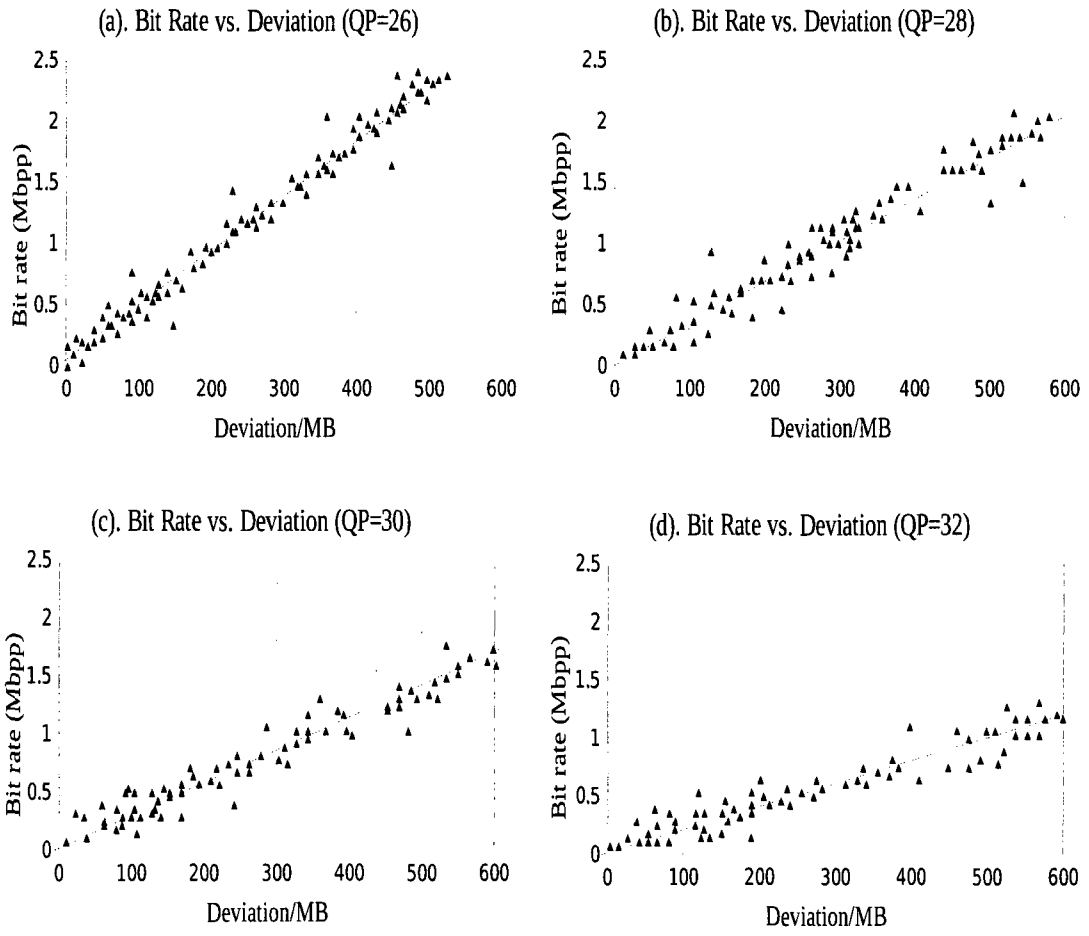


Figure 4.4: Bit rates with different deviation.

4.4.3 Proposed QPs Determination Algorithm and RC Schemes

There are many activity measures or complexity measures for still image coding. Kim [56] proposes four types of measures: discrete cosine transform (DCT)-based, variance-based, gradient-based and edge-based. Hsiz *et* al. [57] proposes a Laplacian of Gaussian (LoG) measure based

on spatial homogeneity to determine the initial QP value. However, the measure only estimates a unique QP for each frame, and thus is not accurate. In our proposed algorithm, four QPs are estimated for each frame based on its spatial features in order to improve the accuracy and video quality.

4.4.3.1 Intelligent Grouping

With the availability of the deviation of each MB, the current intra frame is divided into four groups by the histogram measure as:

$$\sum_{i=0}^{3} \sum_{m=Dev_i}^{Dev_{i+1}} nMB_m = \frac{Height \times Width}{16 \times 16} \tag{4.8}$$

where $i = 0$, 1, 2, and 3, represents the index of the group in the current intra frame. The value Dev_i refers to the start deviation value of the ith group, nMB_m refers to the number of MB, whose deviation value is equal to m, and $Height$ and $Width$ represent the picture's height and width in pixels, respectively. Figure 4.5 illustrates the MB deviation distribution of one frame of "Mobile and Calendar". In this figure, these MBs with equal deviation are gathered into the same bar. The y-axis in this figure indicates the number of MBs while the x-axis indicates the average deviation value of each MB. In the figure, we find that the number of MBs with smaller deviation values is more than that with larger deviation values. As shown in Fig. 4.5, the deviation intervals are equal to 0, 57, 186, and 503, respectively. The MBs with smaller deviation values are assigned into the first group, and MBs with larger deviation values are assigned into the higher groups, accordingly, which means that MBs with smaller deviation values are assigned to smooth regions and thus less bits are allocated while those MBs with larger deviation values are assigned to higher textured regions with more bits allocated.

4.4.3.2 Adaptive Intra R-Q Model

In previous video coding standards such as MPEG-2 [18] and H.263 [19], the QP value is directly used as a scaling factor to control the coding bit rate and the picture quality. This linear relation no longer exists in H.264/AVC standard. Instead, a nonlinear scaling factor

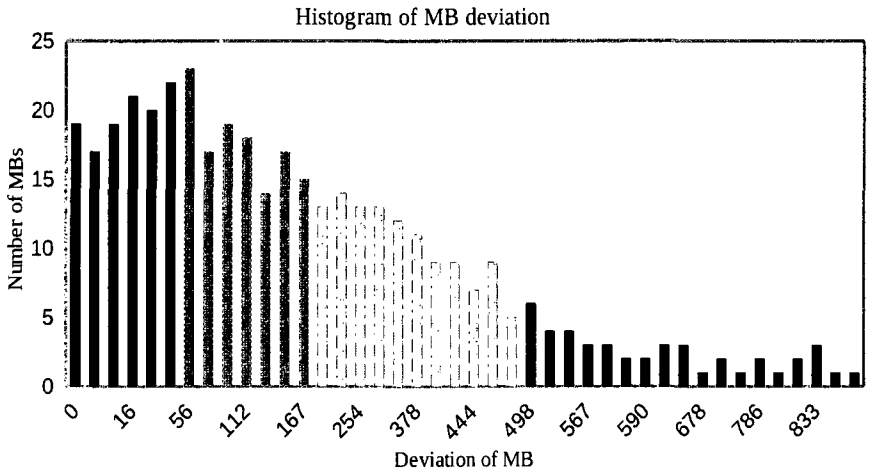


Figure 4.5: Intelligent grouping by deviation.

Q_{step} is used as [47]:

$$Q_{step} = 2^{(QP-4)/6} \tag{4.9}$$

Generally speaking, there are 52 values of Q_{step}, which are indexed by 52 QPs ranging from 0 to 51 and the value of Q_{step} doubles for every increment of 6 in QP [21]. Theoretically, bit rate is more directly related to Q_{step}. Although QP monotonically increases as Q_{step} increases, we model the output bit rate as a function of Q_{step} in our proposed scheme. Experimental observations have shown that the bit rate of intra frame can be formulated by the relations:

$$ItargetBits[i] = (\frac{bit\_rate}{frame\_rate} - Icorr) \times \frac{W[i] \times IntraN[i]}{IntraGOB} \tag{4.10}$$

$$IpredictBits[i] = \frac{SOD_{cur}[i] \times \alpha_{cur}[i]}{Q_{step}[i]} \tag{4.11}$$

where,

$$Icorr = Header\_bits + Iprevious \tag{4.12}$$

$$IntraGOB = \sum_{i=0}^{3} (W[i] \times IntraN[i]) \tag{4.13}$$

$$\alpha_{cur}[i] = \begin{cases} \frac{BR_{pre}[i] \times Q_{step}[i]}{SOD_{cur}[i]} & \text{if (scene changed or initial frame)} \\ \frac{(\alpha_{cur}[i] + \alpha_{pre}[i])}{2} & \text{else} \end{cases} \tag{4.14}$$

where i represents the group number of each frame as in Section 4.4.3.1, bit_rate and $frame\_rate$ are given by the sequence format. The value $Q_{step}[i]$ is obtained by Step 4 of the following proposed R-D model. $ItargetBits[i]$ and $IpredictBits[i]$ indicate the target bit rate and its corresponding prediction bit rate of the ith group, respectively. $Icorr$ is a correction value, which contains header bits of the current frame, $Header\_bits$, and correction bits of its previous frame, $Iprevious$. By experimental observations, we find that the weights of groups, $W[i]$, are better to be assigned to 0.125, 0.225, 0.3 and 0.35 depending on the different importance of each group, respectively. $IntraN[i]$ is the number of MBs of the ith group and $IntraGOB$ represents the sum of the weighted number of MBs of the current frame. The values $\alpha_{cur}[i]$ and $\alpha_{pre}[i]$ are the adaptive weights of luminance for the ith group of the current and previous frames, respectively. Note that $\alpha_{cur}[i]$ has to be re-calculated by using the bit rate value of its previous frame, $BR_{pre}[i]$, as shown in (4.14) when scene change (to be presented in Section 4.5.2) or initial frame occurs because of many content changes in this frame, otherwise, it is obtained by averaging values of $\alpha_{cur}[i]$ and $\alpha_{pre}[i]$. The four initial values for $\alpha_{cur}[i]$ of the intra frame are given by experimental observations. The range of $\alpha_{cur}[i]$ is from 0 to 120. Normally, they are assigned between 35 and 60 initially. Since the values of $\alpha_{cur}[i]$ will be adaptively updated in the following frames so that the selection of the initial values of $\alpha_{cur}[i]$ are not so important. It only affects the updating time. The value $SOD_{cur}[i]$ represents the sum of MB deviations of the ith group in the current frame. The step by step procedure of the proposed R-D model is described as follows:

1. Initialize MaxIndex(= 51), MinIndex(= 0) and CurIndex = MaxIndex, where MaxIndex, MinIndex and CurIndex present maximum, minimum and current QP index, respectively;

2. Calculate the target bit rate for each group using (4.10);

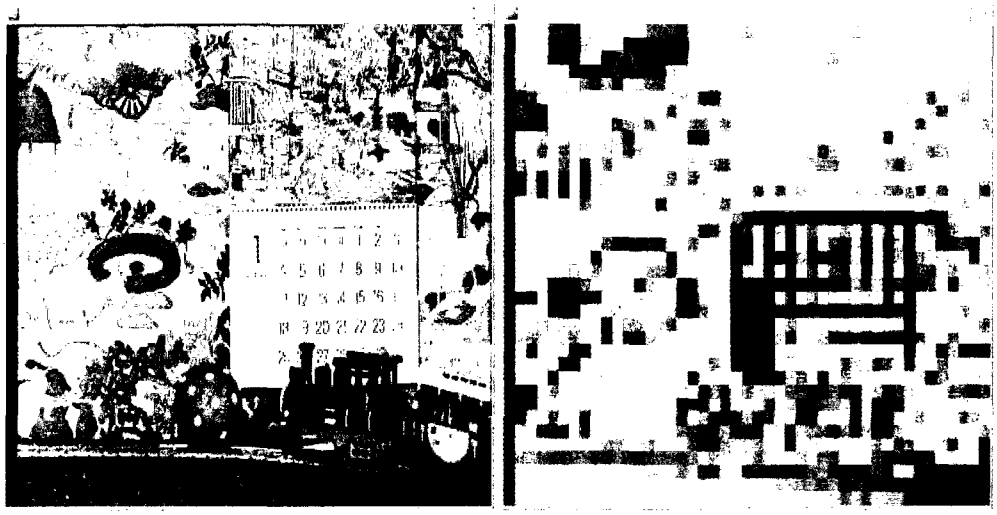
3. Obtain the prediction bit rate for each group using (4.11);

4. QP[i] = int[(MaxIndex + MinIndex) / 2], where int indicates integer value; Find the $Q_{step}[i]$ value using (4.9);

5. If (MaxIndex - MinIndex) $\leq$ 3, QP[i]=CurIndex and terminated, else, go to Step 6;

6. If $IpredictBits[i] > ItargetBits[i]$, MaxIndex = CurIndex, else, MinIndex = CurIndex; Go to Step 4 ;

Figure 4.6 (a) shows an original intra frame of "Mobile and Calendar" and Fig. 4.6 (b) illustrates the QPs distribution of the frame by the proposed R-Q model. In this figure, the four different QP values, 18, 22, 26, and 28, are obtained by the proposed model where they represent four QP values of the four groups of the current intra frame. The smaller blocks in Fig. 4.6 (b) refer to MBs of 16x16 pixels. In the figure, the more white MBs have higher deviation values and thus are assigned to higher QP values while the darker ones have lower deviation values and are assigned to lower QP values.



(a). Original frame. (b). Frame QPs.

Figure 4.6: QPs determination by deviation.

4.4.3.3 Rate Control Schemes

To achieve better performance, we propose two types of encoding schemes to implement the intra frame RC: (1). Even-slice RC, and (2). Priority-slice RC. Both of them follow the same procedure except using different slice partition method. As stated in Section 4.4.3.2, we calculate the deviation of each MB and divide the intra frame into four groups according to the deviation distribution and thus the four QPs are obtained by the proposed R-Q model. Then, we divide the current frame into four super slices. Each super slice includes a quarter

of MB rows of each intra frame. Finally, we implement the encoding process for each super slice. If the encoded bit rate of the current super slice is out of the desired bit rate, a small tuning will be applied to it until the desired bit rate is achieved. The surplus bits of the current super slice are then added to the target bits of the next super slice. By experimental observation, the range of desired bit rate of the first super slice is widely set while that of the last super slice has to be tightly set as much as possible in order to achieve a better RC result. Figure 4.7 illustrates the flowchart of the proposed RC scheme. At the beginning, the RC has to be initialized, some parameters are given by their initial values, such as $\alpha[i] = 48$, where i = 0, 1, 2, and 3, and $Iprevious = 0$. Then, the average deviation of each MB is calculated by (4.3) - (4.7). If the current frame is initial or scene changed frame, the MBs of the frame have to be re-grouped by histogram measure in (4.8), QP values have to be re-calculated by the proposed R-Q model stated in Section 4.4.3.2 and slicing process has to be re-implemented for this frame, otherwise, the parameters of groups, QPs and slices are kept exactly the same as in the previous frame. The different processes of the two types of RC schemes are presented as follows.

Even-slice Rate Control In this approach, the four super slices, $j = 0$, 1, 2, and 3 as shown in Fig. 4.7, are consecutively generated from the first MB row to the last MB row, and each super slice has equal MB rows. The parameter δ is a minimum change based on its original QP[j] values. The main idea is: First, each super slice is encoded with the determined QP[j] value; Second, the encoded bit rate is compared with the target bit rate; Finally, re-encode this super slice with "QP[j]+δ" value if the bit rate consumed are greater than the desired bit rate while re-encoding this super slice with "QP[j]-δ" value if the encoded bit rate is less than the desired one until the encoded bit rate is inside the target bits range.

Priority-slice Rate Control In this approach, the sum of deviation of each MB row is calculated and MB rows are sorted from ones with the larger deviations to ones with the smaller deviations. Each super slice, when $j = 0$, 1, 2, and 3 as shown in Fig. 4.7, contains a quarter of MB rows of an intra frame. The super slices are thus encoded one by one according to their priorities. Super slices with larger deviation values are assigned to higher bit rate

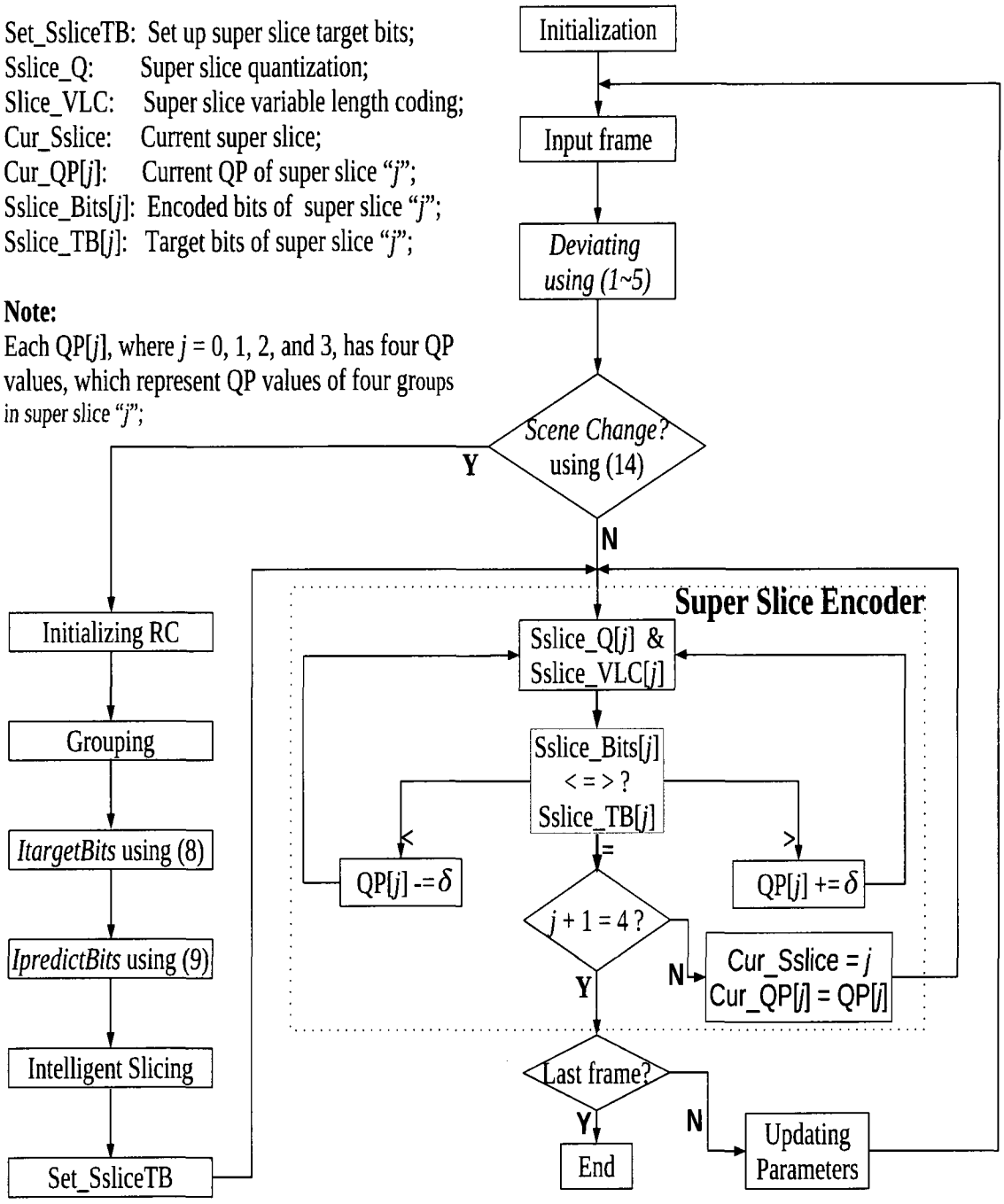


Figure 4.7: Slice rate control.

ranges because they usually have higher output bits fluctuation than the ones with smaller deviations.

4.5 Experimental Results and Analysis

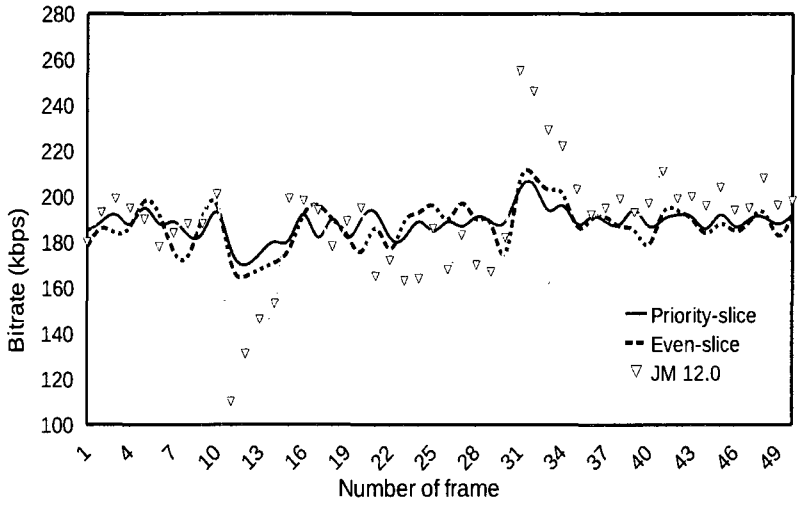
4.5.1 Rate Control Performance

The H.264/AVC reference software JM12.0 [50] is performed to evaluate the proposed algorithms. The baseline profile is selected. In order to evaluate the initial QP determination result, only intra frame is enabled in this experiment. The R-D optimization (RDO) and context adaptive binary arithmetic coding (CABAC) [21] are enabled as well. There are eight benchmark video sequences used in our experiment, which are "Foreman", "Akyio", "News", "Coastguard", "Mobile and Calendar", "Football", "Table tennis", and "Mother and Daughter". Furthermore, to accurately evaluate the proposed RC scheme under scene change, a new sequence with CIF format called "Fancb" is generated since the combined new sequences contain not only eight continuous frames but also eight frames of scene change, which locate in the first frame of each combined sequence. The "Fancb" sequence is formed by combining the first 10 frames of each sequence of "Foreman", "Akyio", "News", "Coastguard" and "Bus". The result of the proposed approaches, aiming target bit rates equal to 192 kbps of "Fancb", are compared to the original RC scheme [45] adopted in H.264/AVC reference model JM12.0 as shown in Fig. 4.8 (a). Obviously, the proposed approaches provide better result, which are more stabler and closer to the target bit rate. On the contrast, the result from JM12.0 with fixed initial QP values is much worse and has higher fluctuations in output bit stream. Figure 4.8 (b) also compares the performance of "Mobile and Calendar" between the proposed approaches and JM12.0 model. With the proposed multiple QPs, a roughly 0.53 dB and 0.74 dB improvements are achieved by the two proposed RC schemes, respectively.

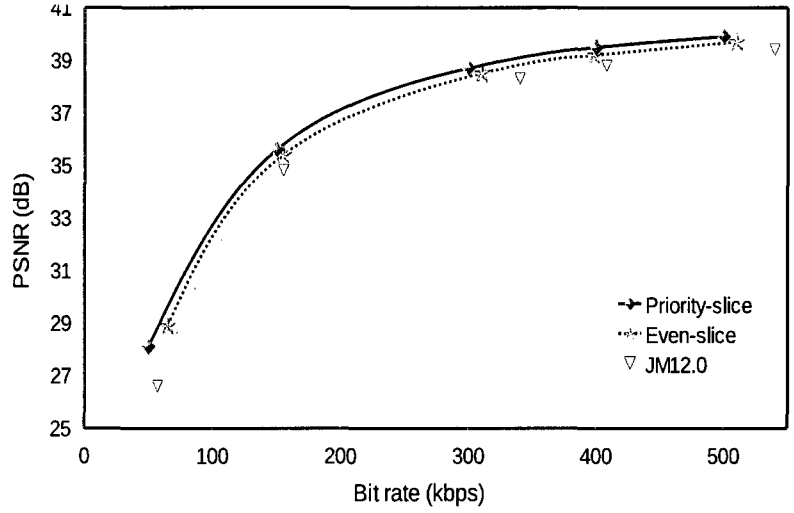
On the other hand, the experiments are also conducted using the first 100 frames of eight sequences to measure the frame bit rate mismatch ratio defined as:

$$M\% = \frac{|R_{target} - R_{actual}|}{R_{target}} \times 100\% \tag{4.15}$$

where R_{target} and R_{actual} are the target bit rate and accurate bit rate, respectively. Table 4.1 tabulates the average value for different sequences. From this table, we can see that the rate estimation accuracy has been greatly improved by the proposed approaches, especially for the



(a). Output bit rate comparison.



(b). Performance (proposed schemes vs. JM12.0).

Figure 4.8: Comparisons of bitrate and performance of "Fancb".

combined sequence. On average, the mismatches of our encoding schemes are about 13.57% and 7.18% compared with 25.15% of JM12.0 model, and we have achieved up to 46% and 71.5% improvements over JM12.0 model.

4.5.2 Scene Change

An abrupt scene transition frame is one that is hardly correlated with the previous frames. In this case because an intra frame has less distortion than an inter frame, almost all MBs

Table 4.1: Performance & mismatch comparison.

Sequences	JM12.0		Even-slice RC		Priority-slice RC	
	PSNR	M%	Δ PSNR	M%	Δ PSNR	M%
Calendar	34.51	19.34	0.55	11.22	0.76	4.31
Coastguard	26.78	32.14	0.46	15.23	0.66	6.78
Foreman	33.62	26.34	0.41	13.32	0.57	8.82
News	34.25	23.11	0.54	15.23	0.67	7.81
Akiyo	29.87	21.43	0.36	11.34	0.55	4.6
Mother	30.89	19.56	0.44	10.12	0.58	5.51
Foot5ball	29.11	31.18	0.76	16.21	1.21	8.98
Table Tennis	33.24	25.13	0.59	14.22	0.82	7.89
Fancb	32.36	55.13	0.66	15.22	0.82	9.89
Average	**31.63**	**25.15**	**0.53**	**13.57**	**0.74**	**7.18**

are encoded in intra mode.However, the RC scheme employed in H.264/AVC adopts quadratic R-D model [45], which is suitable for inter frame, but not for intra frame. It results in more fluctuations of output bit rate and video quality, especially in lower bit rate applications. Therefore, a new RC scheme to improve video quality at scene change is very important. This thesis employs deviation operator to detect scene change and re-calculate QP values for each frame. To detect the scene change frame, the sum of deviation of the frame is used again, which is described by the formula:

$$SceneChange\% = \frac{\sum_{i=0}^{3} |SOD_{cur}[i] - SOD_{pre}[i]|}{\sum_{i=0}^{3} SOD_{cur}[i]} \times 100\% \tag{4.16}$$

where $SOD_{pre}[i]$ refers to sum of MB deviations of the ith group in the previous frame. The selection of threshold value of scene change is a trade-off. If the threshold value is set up too higher, normal scene changes are not able to be detected, and thus the fluctuation of output bit rate happens while lower threshold value of scene change brings with abnormal scene change, and therefore takes long time to update QP values. By experimental observation, a frame with the sum of deviation 25% greater than that of its previous frame can be regarded as scene change, which is suitable for most of scene change cases. To verify the performance of the proposed algorithm in scene change, the mixed sequence "Fancb" is used. Figure 4.9 illustrates the result of the scene change, which would be observed in the transition frame from the sequence "Coast and guard" to the sequence "Bus". Figure 4.9(a) shows the result of H.264/AVC JM12.0 RC and Fig. 4.9 (b) shows the result of the proposed algorithm with priority slicing RC scheme. Obviously, the proposed approach achieves better visual and PSNR performances than the H.264/AVC reference model when scene change occurs.

(a). JM12.0 PSNR=26.3dB. (b). Proposed PSNR=31.7dB.

Figure 4.9: Visual comparison of scene change.

4.6 Conclusions

In this chapter, a novel intra frame QP determination algorithm has been presented. The algorithm adaptively enhances the QP prediction accuracy of the conventional R-Q model by taking the deviation-based frame complexity into consideration. Two types of RC schemes are proposed to reduce output bit rate fluctuations. Also, a scene change detection method is used to reduce degradation of video quality and fluctuations of output bit underflow or overflow. Experimental results have demonstrated that the proposed RC algorithms outperform the reference algorithm employed in H.264/AVC in terms of both performance and fluctuation.

Chapter 5

MPEG-2 to H.264/AVC Transcoding

5.1 Introduction

MPEG-2 [18] has become the primary format for broadcast video after being developed in early 1990's. The new video coding standard, referred to as H.264AVC [21], promises the same quality as MPEG-2 with about half the data rate. Since the H.264/AVC format has been adopted into storage format standards, such as blu-ray disc, we expect H.264/AVC decoders to appear in consumer video recording systems soon. Certainly, as more high-definition content becomes available and the desire to store more content or record more channels simultaneously increases, long recording mode become a key feature for future consumer video recorders. To satisfy this need, we have developed novel techniques that convert MPEG-2 broadcast video to the more compact H.264/AVC format with low complexity. Complexity is kept low by reusing information contained within the MPEG-2 video stream. At the same time, high quality is maintained. The diagram of the proposed system is shown in Fig. 5.1. Since a MPEG-2 decoder is present in existing systems, the challenge is to integrate the simplified H.264/AVC encoding part of the MPEG-2 to H.264/AVC transcoder into the overall system.

Straightforward cascading of a MPEG-2 decoder and a stand-alone H.264/AVC encoder would form a transcoder; This will be referred to as the "reference transcoder" later on in the thesis. The reference transcoder is very computationally complex due to the need to perform

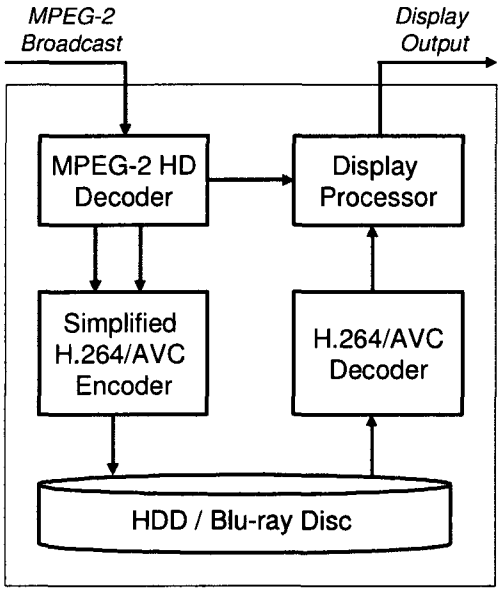


Figure 5.1: Storage system using MPEG-2 to H.264/AVC transcoding.

motion estimation and mode decision in the H.264/AVC encoder. It is well understood that one could reduce the complexity of the reference transcoder by reusing the motion and mode information from the input MPEG-2 video bitstream [58, 59]. However, how such information may be reused in the most cost-effective manner is an open problem. Some existing methods [60–62] analyze the MPEG-2 8x8 DCT DC & AC coefficients of the neighbouring blocks to decide the Intra prediction direction or produce better results using pixel domain transcoding. However these methods could still be computational. Other existing research such as in [63] is on performing the transcoding in the DCT domain for Intra coding, i.e. fast conversion algorithm of the DCT coefficients to integer transform algorithm of the DCT coefficients to integer transform coefficients. Some other fast method using previously coded information is also proposed [64], depending on the activity of the various block size. The existing work [65] introduces a low complexity macroblock partition mode decision algorithm for inter-frame prediction in MPEG-2 to H. 264 transcoder. Some other fast algorithms in [66–69] utilize spatial resolution reduction.

The transcoder architecture we use is shown in Fig. 5.2. It essentially consists of a MPEG-2 decoder and a simplified H.264/AVC encoder. There is a post-processing unit following

the MPEG-2 decoder that may perform artifact removal or resolution scaling if desired. The encoder is "simplified" relative to the reference transcoder, since the motion and mode information is derived based on input MPEG-2 video. In this thesis, we focus on the motion and mode mapping algorithms, the main obstacles in low-complexity transcoder design. We assume the input MPEG-2 video is coded using frame pictures, which is the more popular MPEG-2 coding method. The output will be coded using H.264/AVC frame pictures with macroblock adaptive frame/field (MBAFF) turned off. However, the proposed method could easily be generalized for field picture input and frame picture output with MBAFF or field picture output. In addition, we disable inter prediction for block sizes 8x4, 4x8 and 4x4, although the proposed algorithms can be applied to them too. We think this is a reasonable design for practical applications since block sizes larger than 8x8 are believed to achieve most of the gains promised by variable block size motion compensation.

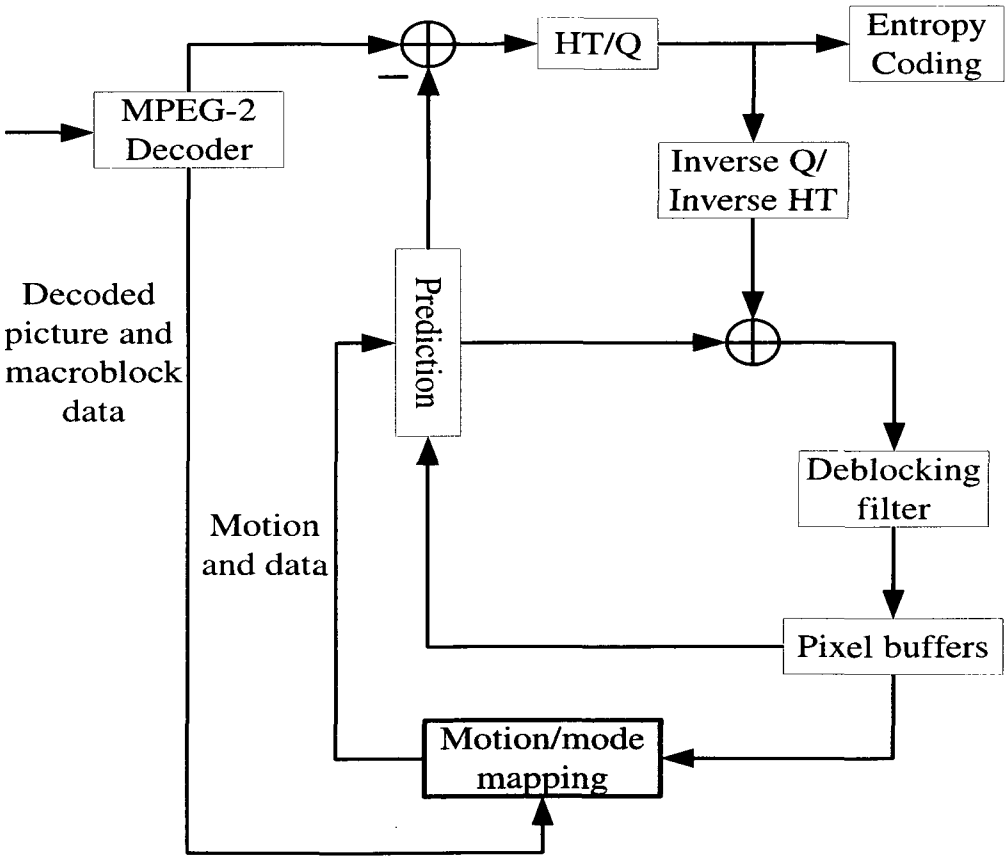


Figure 5.2: MPEG-2 to H.264/AVC transcoding architecture.

In reported works [64, 70] for motion mapping in transcoding, a complete motion estimation algorithm is actually still performed. For inter 16x16 prediction, the motion vectors from incoming MPEG-2 video are used as additional motion vector predictors. For smaller block sizes, e.g. 16x8, 8x16 and 8x8 etc, motion vectors are estimated not directly from incoming motion vectors since MPEG-2 does not have such motion vectors. Instead, these motion vectors are estimated using pure encoding algorithms without considering incoming MPEG-2 motion vectors. Therefore, such an approach still needs very complicated motion search algorithms. In this chapter, we propose a very efficient motion mapping algorithm that directly maps incoming MPEG-2 motion vectors to outgoing H.264/AVC motion vectors, regardless of their supporting block sizes. With the proposed algorithm, the need for complex motion search algorithm is completely eliminated. In addition, we propose an algorithm to support the mapping to H.264/AVC motion vectors with different reference picture, which may be useful in case of picture type conversion or picture skipping, as well as an algorithm to support field to frame motion vector mapping.

For rate-distortion optimized mode decision, the main complexity lies in the Lagrange cost evaluation for all possible coding modes. Existing algorithms typically try to reduce the number of candidate modes actually considered in Lagrange cost evaluation, either based on analysis of pictures for intra mode decision or other heuristics for inter mode decision. In [71], edge direction mapping is used to reduce the number of candidate intra prediction modes. While in [72], edge vector amplitude is used to reduce the number of candidate inter prediction modes. However, these algorithms may be unreliable at times since the pre-analysis may not correlate well with the rate-distortion based decisions. In [73], it is demonstrated that a low complexity cost function can be used to eliminate unlikely inter_4x4 coding modes. The basic idea is to first rank candidate modes in ascending order based on the low complexity cost function. Then the more complex Lagrange cost function is evaluated only for few best modes, i.e. with the lowest costs. In this thesis, we would like to extend the idea so that it would work with inter modes as well. We will show that the low complexity cost function correlates well with the Lagrange cost function. It also provides a level of complexity scalability: we have a low complexity mode decision algorithm if we do not perform the Lagrange cost calculation, and the performance loss is moderate compared to the full algorithm.

The contributions of this work, i.e. the proposed motion and mode mapping algorithms, includes the following:

- A novel motion mapping algorithm directly maps the incoming MPEG-2 motion vectors to outgoing H.264/AVC motion vectors, regardless of their supporting block size, reference picture and frame structure.
- An efficient ranking-based, rate-distortion optimized mode decision algorithm.
- Evaluating effectiveness of various coding tools in the context of transcoding.

5.2 Motion Mapping

The motion mapping algorithm has to solve the following three mismatch problems between MPEG-2 motion vectors and H.264/AVC motion vectors: field/frame mismatch, reference picture mismatch and block size mismatch.

The first type of mismatch is frame/field mismatch. In MPEG-2 frame picture coding, each macroblock can be coded with either frame prediction or field prediction. In frame prediction, a macroblock is predicted from a 16x16 block in the reference frame positioned by a motion vector. In field prediction, a macroblock is divided into two 16x8 blocks, one block belonging to the top field, and the other block belonging to the bottom field. Each 16x8 block has a field selection bit that specifies whether the top or the bottom field of the reference frame is used, and a motion vector that points to the 16x8 pixel block in the appropriate reference field. While in H.264/AVC frame picture coding without MBAFF, only frame prediction is allowed. Therefore, we have to convert incoming MPEG-2 field motion vectors to frame motion vectors.

The second type of mismatch is reference picture mismatch. In case of picture type change or multiple reference picture in H.264/AVC, it is possible that the target motion vector references a different reference picture from the motion vectors available from the input MPEG-2 video.

The third type of mismatch is block size mismatch. H.264/AVC allows use of various block sizes in inter prediction, while there are only motion vectors based on 16x16 blocks from MPEG-2.

Based on the above discussions, we propose a three-step motion mapping approach that apply to each target H.264/AVC block motion vector. We first convert the incoming MPEG-2 field motion vectors (if any) to frame motion vectors, and then map them to reference the same reference picture as the target motion vector. Finally we map the resulted motion vectors to target H.264/AVC motion vectors with the desired supporting block sizes.

After the above motion mapping process is finished, we perform motion refinement centered at the mapped motion vector. First we perform a integer refinement window of ± 1, then we perform half-pel refinement around the best integer motion vector and then quarter-pel refinement around the best half-pel motion vector.

5.2.1 Field-to-Frame Mapping

The following algorithms deal with the case where the incoming MPEG-2 motion vectors for a macroblock are field motion vectors. If a field motion vector refers to the field of the same parity in the reference frame, then it can be directly used as frame motion vector. If a field motion vector refers to the field of the opposite parity in the reference frame, then this motion vector has to be modified.

Without loss of generality, we assume in the video sequence, top field comes first in time. Let us examine the case where the input forward field motion vector is for top field referencing bottom reference field. Based on the assumption that motion is linear over time, we can modify the field motion vector to reference top reference field by linearly scaling the input motion vector. Also notice there is a half pel vertical displacement between top field and bottom field, as illustrated in Fig. 5.3, where the temporal distance between the current frame the reference frame is one frame. In the figure, the input vertical field motion is 0.5, and the output field motion is 2 in field pixel units, and therefore the frame motion is 4 in frame pixel units. For the case of forward motion vector for top field referencing bottom field, the general formula for field to frame mapping is:

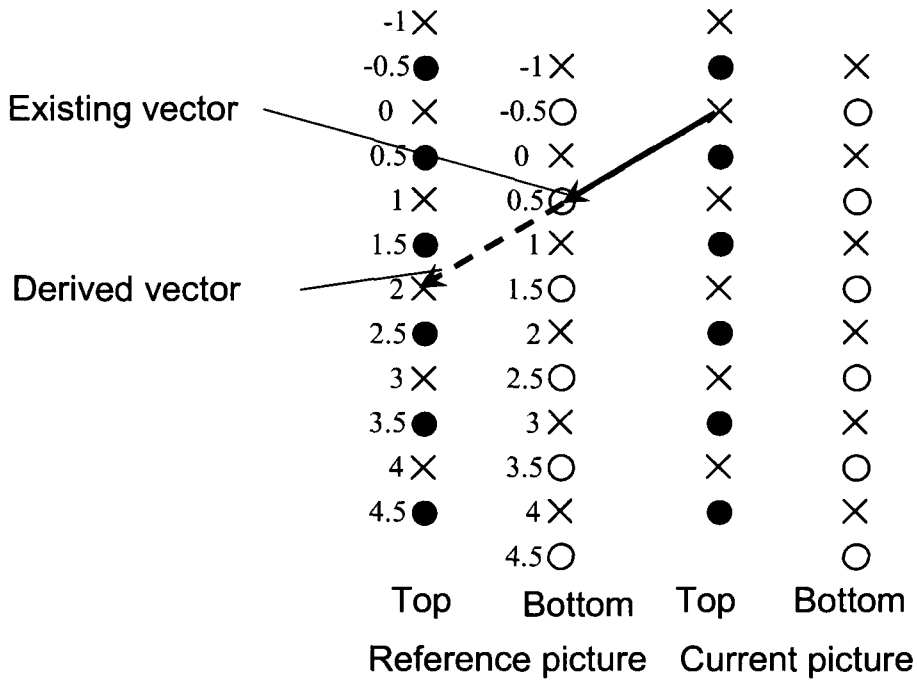


Figure 5.3: Field to frame motion vector mapping.

$$MV_{frame,y} = \frac{2 \times (MV_{field,y} + 0.5) \times (2 \times t_p)}{(2 \times t_p - 1)} \quad (5.1)$$

$$MV_{frame,x} = \frac{2 \times (MV_{field,x}) \times (2 \times t_p)}{(2 \times t_p - 1)} \quad (5.2)$$

where t_p is the temporal distance bewteen current frame and the reference frame. Following the same process, it is straightforward to derive the formula for field to frame mapping for the case of forward motion vector for bottom field referencing top field:

$$MV_{frame,y} = \frac{2 \times (MV_{field,y} - 0.5) \times (2 \times t_p)}{(2 \times t_p + 1)} \quad (5.3)$$

$$MV_{frame,x} = \frac{2 \times (MV_{field,x}) \times (2 \times t_p)}{(2 \times t_p + 1)} \quad (5.4)$$

Similarly, the formula for field to frame mapping for backward motion vectors can be derived. When a macroblock is coded as a field macroblock, it has two field motion vectors, one for top field, and the other one for bottom field. Both of them need to go through the above process. And then the two resulting motion vectors are averaged to form the final mapped frame motion vector.

5.2.2 Reference Picture Mapping

In what follows, we present techniques used to map motion vectors to reference different reference pictures. As an example, we use the case where all non-I pictures are converted to P slices. First, we consider the mapping of P-frame motion vectors as shown in Fig. 5.4.

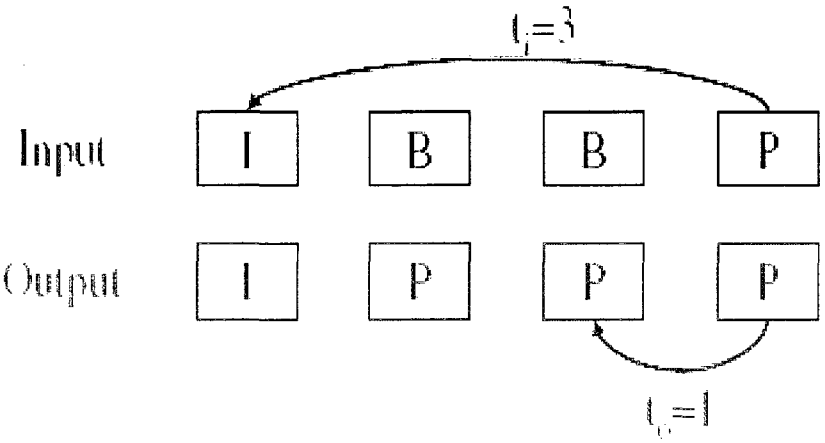


Figure 5.4: Reference picture mapping of motion vector: P to P mapping.

We take the first incoming P-frame as an example. In the input video, the P-frame is predicted from its preceding I-frame. Let t_i be the temporal distance between the P-frame and its reference I-frame. In this example, t_i=3. Assuming only one temporal reference frame is used in the output video, the set of input motion vectors must be modified to reference the preceding P-frame in the output H.264/AVC video. Denote the temporal distance between the output P-picture and its reference P-picture as t_o. In this example, t_o=1. Assuming the motion is small and linear during the period of t_i frames, which is typically 100ms or less, we can represent the mapping from the motion vector of the incoming MB to the output MB

with:

$$MV_o = (MV_i \div t_i) \times t_o \tag{5.5}$$

A more complicated case is shown as the second incoming B frame in Fig. 5.5, where the output is P frame. If a macroblock in the B-frame has a forward motion vector, we use (5.5) to calculate the motion vector for the outgoing P-picture by scaling the incoming forward motion vector. If a macroblock in the B-frame has only a backward motion vector ($MV_{i,back}$), we first add the backward motion vector to the forward motion vector of the collocated future P-frame, i.e., $MV_{i,back} + MV_{i,col}$, and then scale the resulting motion vector according to (5.5).

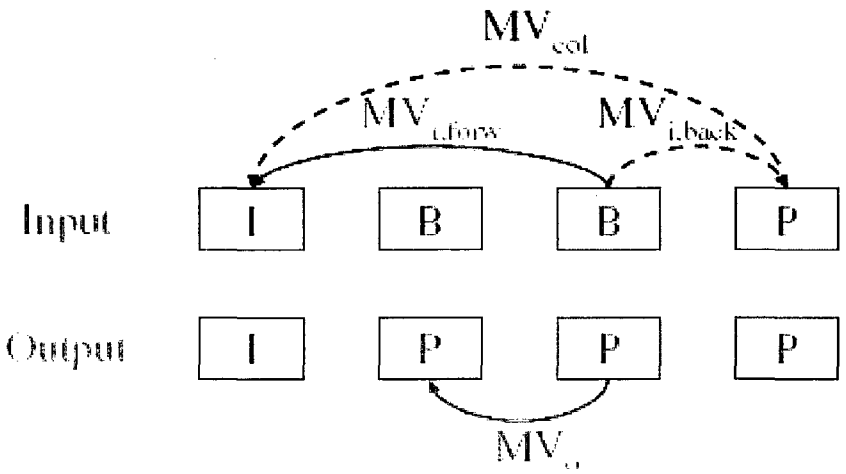


Figure 5.5: Reference picture mapping of motion vector: B to P mapping.

With all incoming motion vectors converted to frame motion vectors and referencing the target reference picture, we can derive the target motion vector using the technique described in Section 5.2.3.

5.2.3 Block Size Mapping

We present two approaches to map motion vectors with 16x16 block size support to motion vectors with smaller supporting block size that will be used in H.264/AVC coding. For target motion vector with 16x16 block support, the input motion vectors can be directly used. The algorithms proposed in this section only apply to target motion vector with supporting block

size smaller than 16x16, i.e. 16x8, 8x16 and 8x8.

5.2.3.1 Distance Weighted Average (DWA)

The assumption of the algorithm is that the motion vector of a rectangular block is same as the motion vector of its geometry center. Then the input to the block size mapping becomes the motion vector of incoming macroblocks' geometry center. And the output becomes the motion vector of the target block's geometry center.

In this algorithm, the output is derived as a weighted average of candidate macroblocks' motion vectors. The candidate macroblocks include current macroblock and those adjacent to the current target block. The weight of an input motion vector is inversely proportional to the distance between its associated macroblock's geometry center to the target block's geometry center.

In Fig. 5.6, for target macroblock partitions A and B for inter_16x8 mode, the candidate macroblocks are labelled with a_1 through a_6, and with b_1 through b_6 respectively. Note that a macroblock has duplicate labels if it is candidate macroblock for deriving both A and B. The geometry centers of each candidate macroblock and target macroblock partitions are also indicated in the figure.

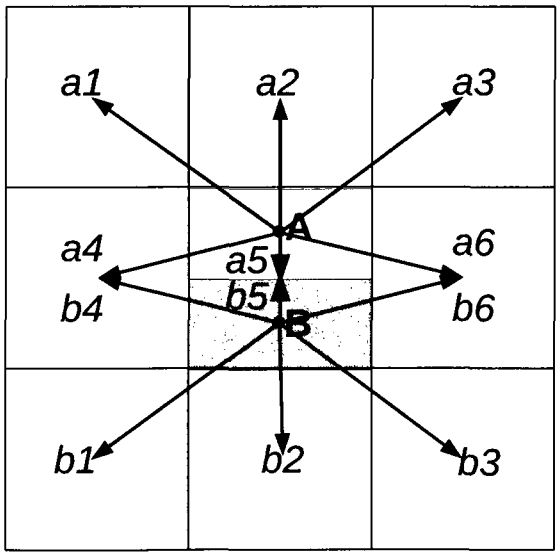


Figure 5.6: Deriving motion vectors for inter_16x8 macroblock partitions with DWA.

Based on the notations given in the figures, here are how the target motion vectors for A

are computed:

$$MV(A) = \frac{\sum_{i=1}^{6} w_i \times MV(a_i)}{\sum_{i=1}^{6} w_i} \tag{5.6}$$

where the weight w_i is proportional to the distance between the geometry center of the candidate macroblock a_i and that of target macroblock partition A. In this case, the values of w_i is given as follows:

$$w_i = \{0.0902, 0.1503, 0.0902, 0.1093, 0.4508, 0.1093\}$$

The motion vector for macroblock partition B can be calculated in a similar fashion using motion vectors of b_i. Similarly, candidate macroblocks are illustrated in Fig. 5.7 and Fig. 5.8 for inter_8x16 and inter_8x8 modes respectively, and the motion vector mapping is performed as distance weighted average of candidate motion vectors.

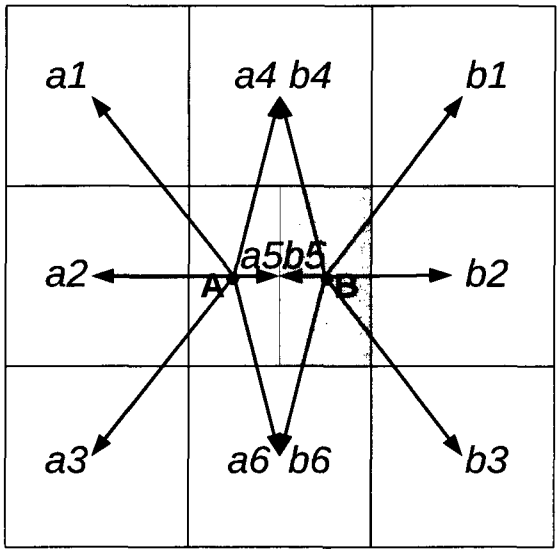


Figure 5.7: Deriving motion vectors for inter_8x16 macroblock partitions with DWA.

Note that the assumption of this approach is more general than translational block motion model typically assumed in related works. Even when there are motions like zoom-in , zoom-out, the motion vector of a rectangular block can be considered to be approximately same as the motion vector of its geometry centre.

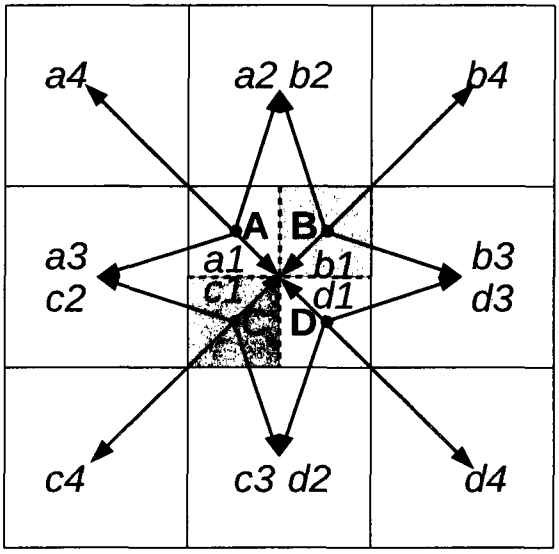


Figure 5.8: Deriving motion vectors for inter_8x8 macroblock partitions with DWA.

5.2.3.2 Error-variance Weighted Average (EWA)

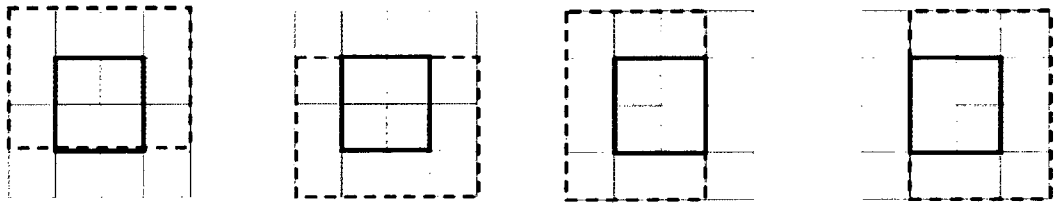
In this algorithm, the output is also derived as a weighted average of candidate macroblocks' motion vectors. The difference from DWA lies in how the candidate motion vectors are formed as well as how the weights are determined. First a mask is applied to each target H.264/AVC motion vector, i.e. each target H.264/AVC macroblock partition for which a motion vector has to be determined. The weighting masks for 16x16, 16x8, 8x16 and 8x8 macroblock partitions are shown as areas enclosed by dashed lines in Fig. 5.9. In these figures, each small square is a 8x8 block. The motion vector of each 8x8 block is assumed to be the vector of the macroblock it belongs to. Actual motion vector mapping is performed per weighting mask.

As shown in Fig. 5.10, each 8x8 block and its motion vector are represented by B_k and d_k, respectively. $x_{i,j}$ denotes the location of a pixel in a 8x8 block, and the prediction error is computed using the following equation from the prediction error for each pixel location $e(x_{i,j}, t)$:

$$e_k = \sum_{(i,j) \in B_k} |e(x_{(i,j)}, t)| \tag{5.7}$$

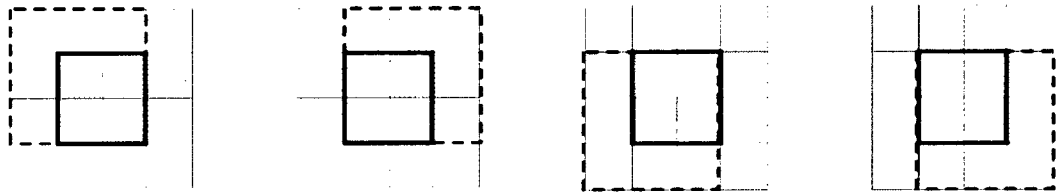
And, the variance of th motion vectors within a mask is derived as follows:

$$\sigma = Var(d_i), i \in R \tag{5.8}$$



(a). Inter 16 x 8 partition.

(b). Inter 8 x 16 partition.



(c). Inter 8 x 8 partition.

Figure 5.9: EWA weighting masks.

where, R is the weighting mask. Finally, H.264/AVC Inter_8x8 motion vector is obtained through:

$$\hat{d} = \frac{\sum_R \varphi_k d_k}{\sum_R \varphi_k} \qquad \varphi_k = \begin{cases} \frac{A}{e_k} & k \neq 4 \\ \frac{A}{h(\sigma) e_k} & k = 4 \end{cases} \tag{5.9}$$

In (5.9), A is a constant value and $h(\sigma)$ specifies a monotonically-decreased discrete function of σ (various of motion vectors in a mask). As a result of this process, we have the mapped motion vector for each H.264/AVC target macroblock partition.

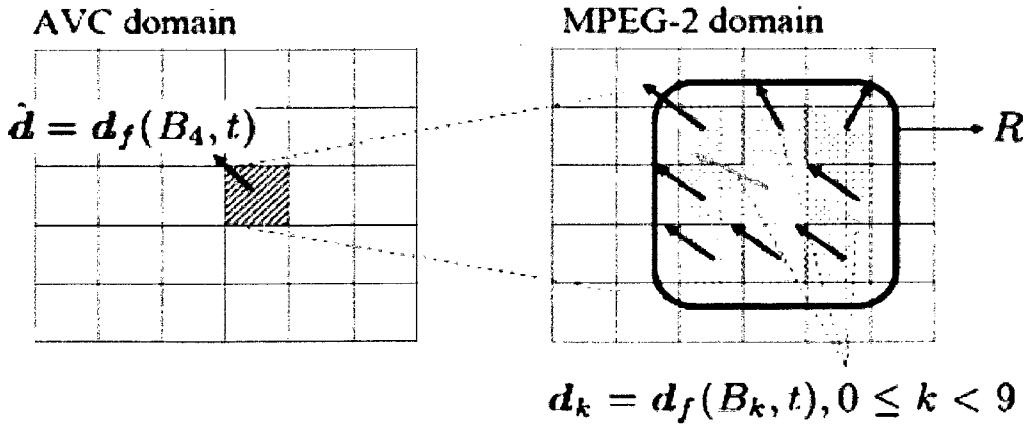


Figure 5.10: Error-variance weighted mapping for inter_8x8 block.

5.3 Mode Decision

In H.264/AVC, a macroblock can be coded in one of many possible modes. It is therefore very important to have a cost-effective mode decision algorithm, otherwise some of coding efficiency benefits of H.264/AVC coding may not be realized. We briefly review the rate-distortion optimized (RDO) mode decision algorithm used in JM reference software below. The goal of RDO mode decision is to help select the best block partitioning as well as inter/intra decision for a macroblock. First, the coding rate and the resulting distortion are computed for all possible macroblock coding modes. Then the Lagrange cost function is evaluated for each mode m:

$$J_1(m) = SSD(m) + \lambda_{mode} \times R(m, Q); \tag{5.10}$$

where, $SSD(m)$ is the distortion, i.e. the sum of squared distance bewteen the original block, s, and the reconstructed block using mode m, $\tilde{s}(m)$ is expressed as:

$$SSD(m) = \left\| s - \tilde{s}(m) \right\|_2^2 \tag{5.11}$$

where, $\| \cdot \|$ is L_p-norm. In (5.10), $R(m, Q)$ is the total number of bits used to code the macroblock, including overhead such as mode and motion vectors, as well as transform coefficients. λ_{mode} is the Lagrange multiplier that controls the rate-distortion trade off. Q is the quantizer used for quantization of transform coefficients. As used in the JM software, λ_{mode} is set according to H.264/AVC quantization parameter as follows:

$$\lambda_{mode} = 0.85 \times 2^{QP/3} \tag{5.12}$$

The optimum mode is the one that minimizes the Lagrange cost function:

$$m^* = argmin_{m \in M} \left(J_1(m) \right) \tag{5.13}$$

where, M is the set of candidate modes. As an example, for P-slices in H.264/AVC, the set of candidate macroblock modes are {skip, inter16x16, inter16x8, inter8x16, inter8x8, intra4x4, intra16x16}.

5.3.1 Ranking Based Mode Decision

The process for calculating the Lagrange cost needs be performed many times since there are a large number of available modes for coding a macroblock. Therefore, the computation of the rate-distortion optimized coding mode decision is very intensive. This motivates us to develop a more efficient mode decision algorithm. The basic idea is to rank all candidate modes using a simpler cost function, and then evaluate the more complex Lagrange rate-distortion costs only for the few best modes determined by the ranking. The simpler cost function we choose to use is the low-complexity cost function used in JM software based on the sum of absolute distance of the Hadamard-transformed prediction residual error signal:

$$SATD(m) = \left\| T(s - \hat{s}(m)) \right\|_1 \tag{5.14}$$

where $\hat{s}(m)$ is the prediction signal using the mode m, and $T()$ is the Hadamard-transform operator. The cost function would then be given by

$$J_2(m) = SATD(m) \tag{5.15}$$

So first we compute the SATD cost $J_2(m)$ for all possible modes. Then we sort the modes according to their SATD costs in ascending order, and put the first k modes in the a test set, denoted as T. For the modes in T, compute the Lagrange cost J_1 using (5.10), which is much complexer than computing the cost J_2 with SATD using (5.15), and then we finally select the best mode according to the Lagrange cost. The parameter k controls the complexity-quality trade-off. To verify the correlations between rankings using SATD cost and Lagrange cost, a simple experiment is performed. We collect the two costs for all luma 4x4 blocks in the first frame of 5 CIF test sequences coded with QP=28, and then count the percentage of times when the best mode according to Lagrange cost is in the test set T. This is called the mode prediction accuracy. The results are plotted in Fig. 5.11 as k vs. mode prediction accuracy. The strong correlation between the two costs is evident in the high accuracies shown. In this work, k is set to be "3".

Note that the above algorithm applies first to decisions of Intra4x4 prediction modes, and

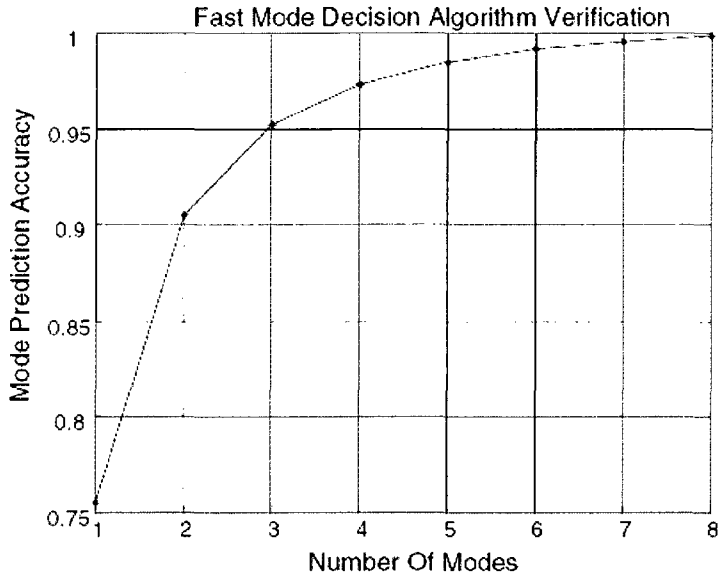


Figure 5.11: Number of test modes vs. accuracy.

then again to the decisions of various inter and intra prediction modes. And when applied to Intra4x4 prediction modes, the SATD cost for for has an additional term compared to 5.15.

$$J_2(m) = SATD(m) + \lambda_{mode} \times 4 \times (1 - \delta(m = m^+)) \tag{5.16}$$

where m^+ is the most probable mode for the block. The additional term is to add a simple rate constrain.

5.3.2 Transform Domain Cost Calculation

Transform domain cost calculation is based on our previous work [74], where we found out that the coding distortion, Sum of Squared Distance (SSD) can be more efficiently calculated in transform domain without reconstruction of pixels as follows.

$$SSD(m) = \left\| (E - \tilde{E}(m) \otimes W_1) \otimes W_2 \right\|_2^2 \tag{5.17}$$

where E and $\tilde{E}$ are the transformed residual signal and reconstructed transform residual signal through inverse scaling and inverse transform. $\otimes$ is the operator for scalar multiplication or element-wise multiplication. W1 and W2 are weighting matrices to compensate for the

different norms of H.264/AVC transform and quantization design, and are given as:

$$W_1 = \frac{1}{64}\begin{pmatrix} 10 & 20 & 16 & 20 \\ 20 & 25 & 20 & 25 \\ 10 & 20 & 16 & 20 \\ 20 & 25 & 20 & 25 \end{pmatrix} \tag{5.18}$$

$$W_2 = \begin{pmatrix} \frac{1}{4} & \frac{1}{\sqrt{40}} & \frac{1}{4} & \frac{1}{\sqrt{40}} \\ \frac{1}{\sqrt{40}} & \frac{1}{10} & \frac{1}{\sqrt{40}} & \frac{1}{10} \\ \frac{1}{4} & \frac{1}{\sqrt{40}} & \frac{1}{4} & \frac{1}{\sqrt{40}} \\ \frac{1}{\sqrt{40}} & \frac{1}{10} & \frac{1}{\sqrt{40}} & \frac{1}{10} \end{pmatrix} \tag{5.19}$$

When we can compute the distortion in transform domain, there is not need to perform the inverse transformation and reconstruction of pixels. Therefore complexity for mode decision is reduced. This technique is used in calculation the Lagrange rate-distortion costs in the intra4x4 mode decision process. For more details about the technique, please refer to [74]

5.4 Simulation Results

We use two interlaced sequences in the simulations: HarborScene and StreetCar. Both of them have resolution 1920×1080 with frame rate 30 frames/s, and are 15 seconds long (450 frames). They are encoded using the MPEG-2 reference software [75] at 30 Mbps and are used as input to the transcoder. The group of picture (GOP) size for MPEG-2 encoding is N=30 and two B-frames are coded between every consecutive pair of anchor frames, i.e. M=3. The search ranges are set to be 85, 170, 255 for temporal prediction distance of 1, 2 and 3 frames, respectively.

As a benchmark, we compare the performance of the proposed transcoding algorithms to the reference transcoder, which is actually a cascaded MPEG-2 decoder and a H.264/AVC encoder, where the H.264/AVC encoder encodes the pictures reconstructed by the MPEG-2 decoder. The H.264/AVC encoder we use is the JM10.2 reference software [38]. Inter predictions using block sizes 4x8, 8x4 and 4x4 are disabled to make fair comparisons. In all simulations, I-frames are always transcoded to I-pictures. The QP values are chosen for the

H.264/AVC quantization such that the output bit rate is around 10 Mbps, which is the target bit rate of interest for consumer storage applications. UVLC is the H.264/AVC entropy coding method. The rate-distortion plots that evaluate various aspects of the proposed transcoder are shown from Fig. 5.12 to Fig. 5.15.

The relative complexities for proposed transcoder algorithms are shown in Figure 5.12. The complexity numbers are measured as the average CPU time consumed by the transcoder for all QP values for the StreetCar sequence. Only StreetCar results are shown since the results are similar for other sequences. Complexity analysis will be provided in further discussions below.

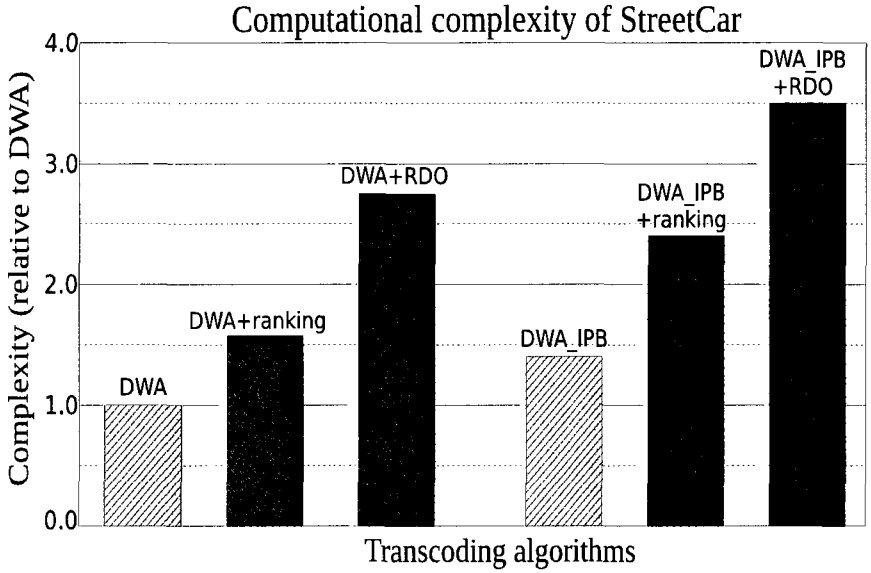


Figure 5.12: Transcoding complexity.

We use four interlaced sequences in the simulations: HarborScene, StreetCar, SoccerAction, and HorseRace. All of them have resolution 1920x1080, frame rate 30 frames, and length 15 seconds (450 frames). They are encoded using the MPEG-2 reference software [75] at 30 Mbps and are used as input to the transcoder. The group of picture size for MPEG-2 encoding is 30, and two B frames are coded between every consecutive pair of anchor frames, i.e. I and P frames. (We pick 30 Mbps since it is not an optimized encoder and the quality at 30 Mbps was found to be comparable visually to broadcasting content.)

As a benchmark, we compare the performance of the proposed transcoding algorithms to

the reference transcoder, which is actually a cascaded MPEG-2 decoder and a H.264/AVC encoder, where the H.264/AVC encoder encodes the pictures reconstructed by the MPEG-2 decoder. The H.264/AVC encoder we use is the JM10.2 reference software [38] with the following parameters: Universal Variable Length Coding (UVLC), and UseFME=1. Inter predictions using block sizes 4x8, 8x4 and 4x4 are disabled to make the comparison fair.

In all simulations, I frames are always transcoded to I pictures. The QP values are chosen for the H.264/AVC quantization such that the output bit rate is around 10 Mbps, which is the target bit rate we are interested to achieve.

5.4.1 Motion Mapping Evaluation

The simulation results reported in this subsection are aimed at evaluating the performance of the DWA and EWA motion mapping algorithms. We simulate two transcoders, one using the motion mapping process described in Section. 5.2 with DWA, and the other one with EWA. We compare the performance of the two transcoders to that of the reference transcoder. To demonstrate the performance of the complete mapping algorithm, we convert incoming P- and B-frames to P-pictures of H.264/AVC output. This effectively simulates the case in which compliance to the H.264/AVC Baseline Profile is desired.

The results are shown in Fig. 5.13. It is clear from these plots that the proposed mapping algorithms (both DWA and EWA) achieve comparable rate-distortion performance to the reference transcoder. At 10 Mbps rate point, the performance loss relative to the reference transcoder in terms of PSNR is less than 0.4 dB for HarborScene and about 0.15 dB for StreetCar. Compared to the reference transcoder using exhaustive search, the complexity saving is more than 95%. The complexity is measured using consumed CPU time, and the computational saving is similar for both sequences and methods. Both DWA and EWA have almost the same PSNR performance (hard to tell the difference from the rate-distortion plot), therefore in all remaining simulations we will use DWA as the mapping algorithm. In terms of computational complexity, DWA is slightly simpler than EWA. However, the difference is fairly small considering the overall transcoder complexity. For this reason, in the complexity plot Fig. 5.12, only DWA complexity is shown.

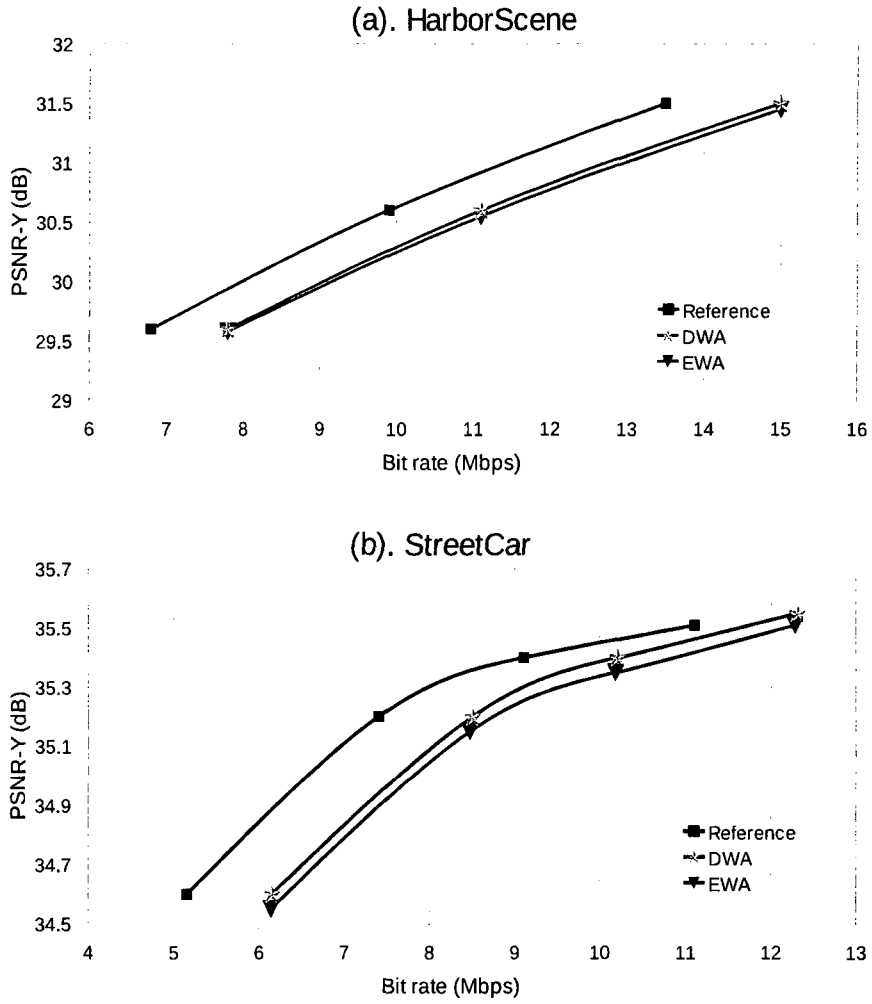


Figure 5.13: Motion mapping performance.

5.4.2 Mode Decision Evaluation

The simulation results reported in this section evaluate the performance of the proposed ranking-based mode decision algorithm. We simulate two transcoders with RDO turned on. One uses the proposed ranking-based RDO algorithm, while the other one uses an exhaustive RDO algorithm. Both use the DWA mapping algorithm. As additional points of comparison, we also plot the performance of the DWA mapping with RDO off (DWA) and the reference transcoder with RDO on (REF+RDO).

As shown in Fig. 5.14, the performance of the simplified ranking-based mode decision (DWA+ranking) is very close to the RDO mode decision (DWA+RDO), with negligible PSNR

loss of less than 0.1dB. The simplified mode decision saves close to 50% of the computation compared to the exhaustive RDO mode decision. We also observe that the simplified RDO mode decision adds about 50% complexity relative to DWA without RDO, while achieves a small PSNR improvement. Consistent to the previous set of simulations, small performance gaps can be observed relative to the reference transcoder with RDO.

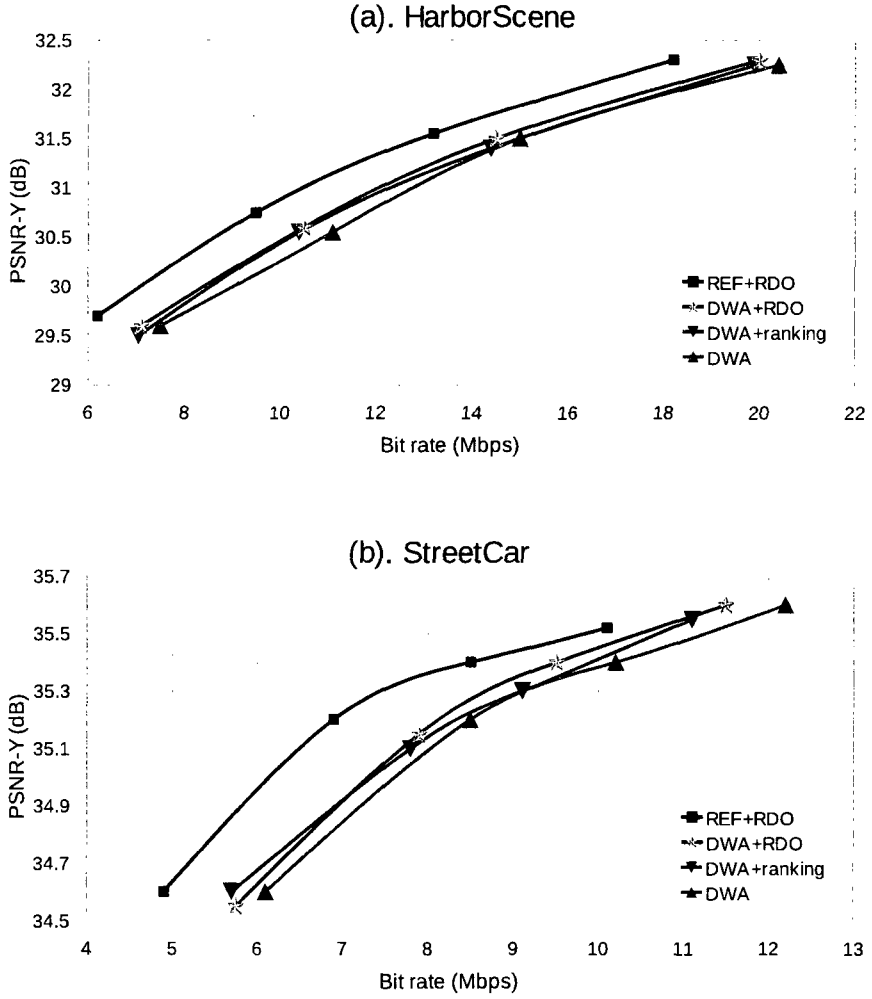


Figure 5.14: RD modes ranking performance.

5.4.3 Impact of B Slice

In the last set of simulations, we examine the performance changes when B-slices are introduced as part of the output. In Fig. 5.15, performance of proposed transcoder using DWA and B

slices with RDO both on (DWA_IPB+RDO) and off (DWA_IPB) are shown. The performance of DWA without RDO or B slices are shown as reference. We also plot the performance of reference transcoder using B slices and RDO (REF_IPB+RDO) for comparison. The interesting point worth noting is that introducing B-slices actually improves the compression efficiency fairly significantly while not significantly increasing the complexity. The reason seems to be that when B-slices are introduced, the number of frames for which sub-pel interpolation is needed is reduced by 2/3. Although transcoding B-slices would need more complexity for motion compensation and motion mapping, the complexity saving from sub-pel interpolation offsets much of the increase. This suggests B-slices could be a more cost-effective tool for the transcoder design than the RDO tool. We can also see from the plots that the proposed transcoder achieves comparable quality to the reference transcoder.

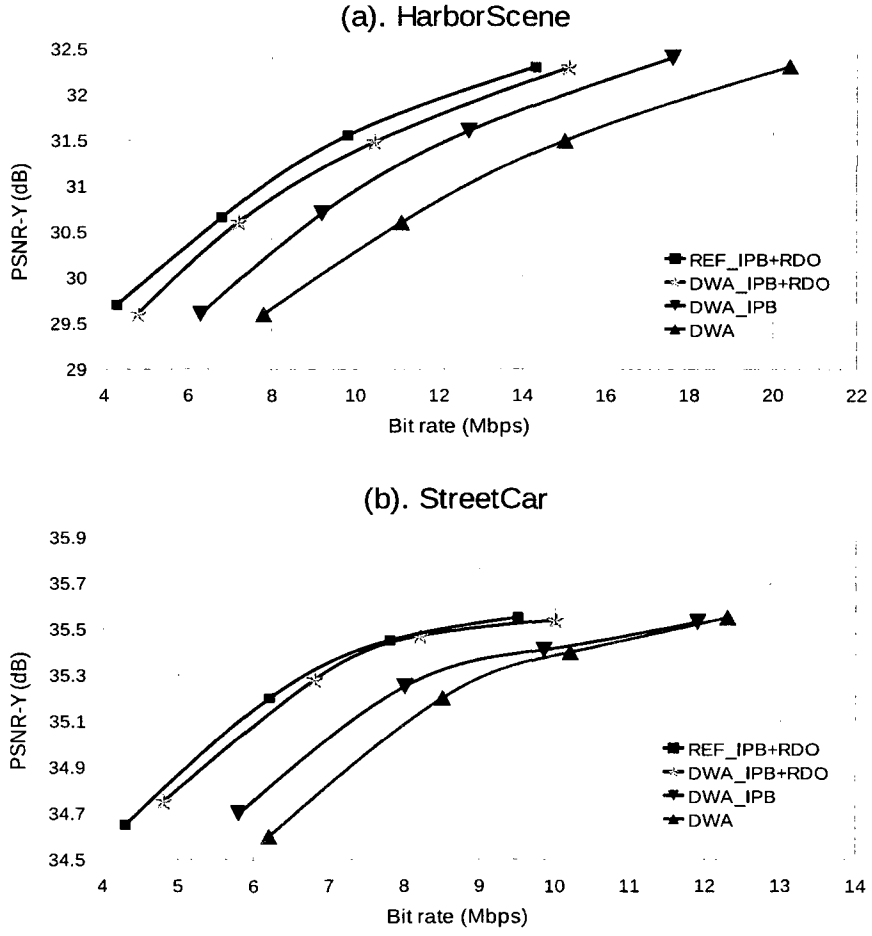


Figure 5.15: Motion mapping with B performance.

5.4.4 Performance Comparison

Table 5.1 shows a comparison of the actual running time on a Pentium-IV 2.66 GHz between the cascade transcoder and proposed transcoder using an arbitrary QP value. The proposed methods are 3–4.3 times faster than the cascade transcoding method in the case of the HD1080 sequences, respectively. This table aslo shows the H.264 bitrates, PSNR, and actual running time of the transcoders for various sequences for the given QP values.

Table 5.1: Performances comparison.

Sequence	**QP**	**Experiment**	**PSNR** (dB)	**Bit Rate** (Mbits)	**Time** (Secs)
WhaleShow	27	Cascading	29.12	121.3	3.13
		Transcoder	28.12	111.2	1.14
	30	Cascading	31.12	135.1	3.53
		Transcoder	28.12	127.4	1.47
	33	Cascading	34.12	142.3	3.91
		Transcoder	28.12	137.5	1.84
EuropeanMarket	27	Cascading	28.42	101.22	2.21
		Transcoder	27.14	89.34	0.83
	30	Cascading	30.72	113.45	2.81
		Transcoder	29.92	100.34	0.89
	33	Cascading	34.12	121.45	3.13
		Transcoder	33.12	109.1	1.07
StreetBus	27	Cascading	30.62	141.12	3.73
		Transcoder	29.24	113.21	0.91
	30	Cascading	33.02	151.45	3.93
		Transcoder	32.52	131	1.19
	33	Cascading	35.32	167.23	4.11
		Transcoder	34.42	141.98	1.43

5.5 Conclusions

We presented motion mapping algorithms that can efficiently map incoming MPEG-2 motion vectors to outgoing H.264/AVC motion vectors, even when they have different block size support and different reference pictures. We then presented an efficient rate-distortion optimized macroblock coding mode decision algorithm, where we first evaluate candidate modes based on a simple cost function so that a reduced set of candidate modes is formed, then we evaluate the more complex Lagrangian cost calculation only for the reduced set of modes. Extensive simulation results show that our proposed transcoder incorporating the proposed algorithms

could achieve good rate-distortion performance with low complexity. Compared with cascaded decoder-encoder solution, the RD performance is maintained while the complexity is significantly reduced. We also compared the transcoding performance with and without frame type conversion.

With the above transcoding algorithms, the current transcoder's complexity is dominated by sub-pel interpolation and motion refinement. And mode decision still has considerable complexity. Therefore, it will be interesting to further explore along these directions to have more efficient transcoder design without affecting the compression efficiency.

Chapter 6

Efficient Dataflow VLD Implementation for MPEG-4 SP RVC Framework

6.1 Introduction

Nowadays, video decoders need to support multiple codec standards because more and more video standards are deployed. Although different, all coding standards use the same or very similar coding tools and results to share similar architectures and implementations. Unfortunately, the way in which the existing coding standards are specified lacks of flexibility to adapt performances and complexity when new applications emerge. MPEG reconfigurable video coding (RVC) standard intends to create a framework containing existing coding technology for developing, beside current standard decoders, new configurations for satisfying specific application constraints. RVC introduces a novelty since it promotes standardization solutions from different implementers. One challenge posed by the possibility of reconfiguring decoders is the need of appropriate procedures for the instantiation and synthesis of bit stream parsers in which efficient variable length decoding processes are important tasks. This thesis presents a method for generating efficient components for the MPEG RVC library capable of decoding variable length codes. The components of the library like all other coding tools are CAL actors generated automatically given the input VLD table. By using the described procedure, VLD

tables can be automatically and efficiently generated as FUs of RVC toolbox. By efficiently it is also meant that the data flow CAL FUs are suitable for efficient synthesis into software (SW) and hardware (HW) implementations.

6.2 MPEG Reconfigurable Video Coding Overview

MPEG has always worked to propose innovations in the video coding field that are capable of satisfying the changing landscape and needs of video coding applications. With this objective, MPEG intends to standardize the reconfigurable video coding framework allowing a dynamic development, implementation and adoption of standardized video coding solutions based on a unified library of components with features of higher flexibility and reusability. RVC is a flexible framework for MPEG that tries to provide a systematic way of constructing video codecs from a collection of coding tools, it has been firstly presented in [76]. The goal of the introduction of such new interoperable model at coding tool level is twofold: to speed up the adoption and standardization of new technologies by adding new tools in toolbox and to enable the dynamic definition of new profiles. The modular data flow based specification formalism also provides a starting point for design that is adapted to yield direct synthesis of SW and HW by using appropriate tools, for direct mapping on SW and HW platforms.

A decoder specification under RVC is defined with the standard MPEG toolbox (instantiation and connections of the different coding tools) and the specification of the video bit stream syntax expressed in a MPEG-21 bit stream syntax description language (BSDL) schema [77]. The toolbox consists of various coding tools which are also named functional units (FU). Each FU is a modular coding tool (such as inverse discrete cosine transform (IDCT), motion compression (MC)).

The concept of RVC framework can be illustrated by Fig. 6.1. The key difference between RVC and traditional codec standards is their conformance point. The traditional codec standards define their conformance point at decoder level whereas RVC defines it in tools level so that RVC enables much more flexibility and several configurations of components taken by previous monolithic specifications are possible.

Another fundamental difference between RVC specification and the traditional standard codec specification is the data flow based formalism. In the traditional codec specifications,

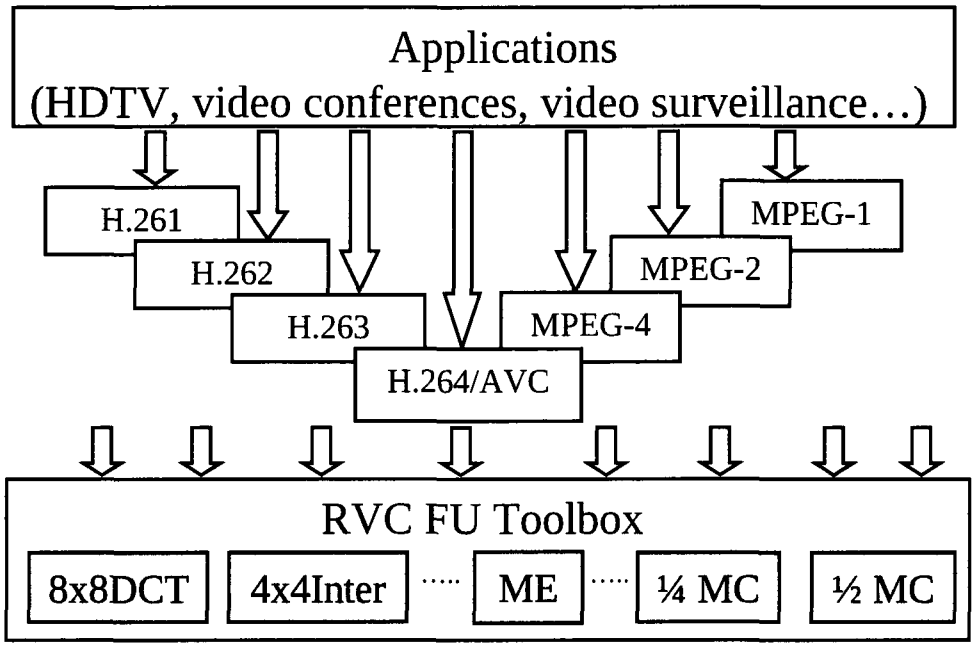


Figure 6.1: RVC framework.

C/C++ is the language of the reference SW, which is usually composed by several thousands of lines and is getting more and more difficult to understand and to transform into efficient implementations. In the RVC framework, data flow actor-oriental language CAL which is simpler, compact in terms of number of code lines, and does not include non necessary implementations details such as a fixed scheduling for C/C++ reference SW, for instance, is used to describe FUs behavior.

6.3 Variable Length Decoding for the RVC Framework

One problem that needs to be solved when applying RVC is how to specify the parser that is in charge of decoding the bit stream of compressed video. In fact, whereas all FUs of the standard MPEG toolbox are available under the form of CAL actors or as a proprietary implementation for specific platforms, the parser of a new decoder configuration need to be synthesized and instantiated automatically because it is a too burdensome task to let the designer write the parser actor in CAL. The parser is not considered as a coding tool because it does not contains any algorithm described by the standard. The unique task of the parser is to feed the coding tools with the right coded data contained in the bit stream. Therefore, a systematic procedure

for synthesizing efficient parsers using appropriate FUs available in the standard toolbox is required.

6.3.1 Solution for Variable Length Decoding

Variable length coding is the most popular entropy coding module which is used in many video and picture coding standards, such as JPEG, MPEG-x, and H.26x. One of the difficulties for RVC to describe variable length decoding is the large amount of tables. For example, in MPEG-4 SP [78] there are 8 tables and in MPEG-4 ASP [78], there are 19 tables. Including those tables directly in the syntax description (BSDL schema transmitted as header in the bit stream) would imply inefficiency in the compactness of the description of a new codec configuration, but would also requires large memory and bandwidth. Another difficulty is the parsing process of the undefined bit length of syntax. In order to avoid carrying VLD tables in bit stream description, VLD tables could be separated and implemented in CAL as FUs of RVC toolbox. The proposed Huffman decoding method is applied to VLD tables, which further improves efficiency. The bit stream syntax parser is generated automatically as an independent FU in CAL language from an extensible markup language (XML) schema describing the structure of the bit stream. The transformation process is implemented using extensible style sheet language transformations (XSLT). The bit stream schema is specified in BSDL [77], a MPEG-21 standard. Negotiation between the syntax parser and VLD tables are also established in XSLT for variable decoding process. The systematic solution for syntax parser is highly efficient and flexible to decode a reconfigured bit stream.

6.3.2 Efficient Huffman Decoding Method

In this section, a CAL model for efficient Huffman decoding is proposed for VLD tables of MPEG-2 and MPEG-4. The proposed implementation is optimized aiming at searching time and memory requirement reduction. Huffman coding has been adopted by MPEG-2 and MPEG-4 entropy coding. Sets of codewords are defined based on the probability distributions of "generic" video material. The direct way to decode variable length syntax is using a full search method:

1. The variable length decoder receives one bit from bit stream.

2. Look through the corresponding table from the beginning to check whether it is coincide with certain code.

3. If it is found, output the value from the table.

4. Or else receive another bit and combine it with the former bits, go back to Step 1.

Such full search method is simple but not efficient enough because of duplicate lookup once one bit is received. In addition, it requires a 2-D memory for each table which is not a good choice for hardware implementation. The proposed method rearranges the code in the Huffman tree. The binary Huffman tree search can find the optimal route in short time and requires less medium data. As shown in Fig. 6.2, the variable length coding codeword starts with the first incoming bit. The current bit goes to the left leaf if the coming bit is "0". Otherwise, it goes to the right leaf while "1". Weight of each leaf is marked with the same value of lookup index for corresponding VLD tables. That is to say, every time one bit is consumed at input, one index is generated and one lookup result is generated as output. If the result is a true decoded value, it is provided to the output of the CAL FUs and the search of the variable length coding is completed. On the other hand, if the result is a false decoded value, a further searching is continued until a completed codeword is found.

Different video coding standards have different VLD tables. Even in a single standard, different profiles and levels have different VLD tables' scope. The most efficient solution for the RVC framework would be to build separate FUs available in the standard toolbox for each VLD table decoding and then generate dynamically a parser as composition of a synthesized CAL parser and VLD decoding FUs.. Each VLD table is considered as an independent FU of the RVC toolbox. For example, in MPEG-4 specification Annex B, there are 8 VLD tables that are used by a simple profile decoder. They are B-6, B-7, B-8, B-12, B-13, B14, B-16 and B-17. In the MPEG-4 advanced simple profile, to these tables other VLD tables are needed. It is unnecessary to generate them once again, just access to toolbox and get related FUs. Take MPEG-4 SP for example, we generate the VLD FUs and name them with the table name, such as B-6, B-7 and so on.

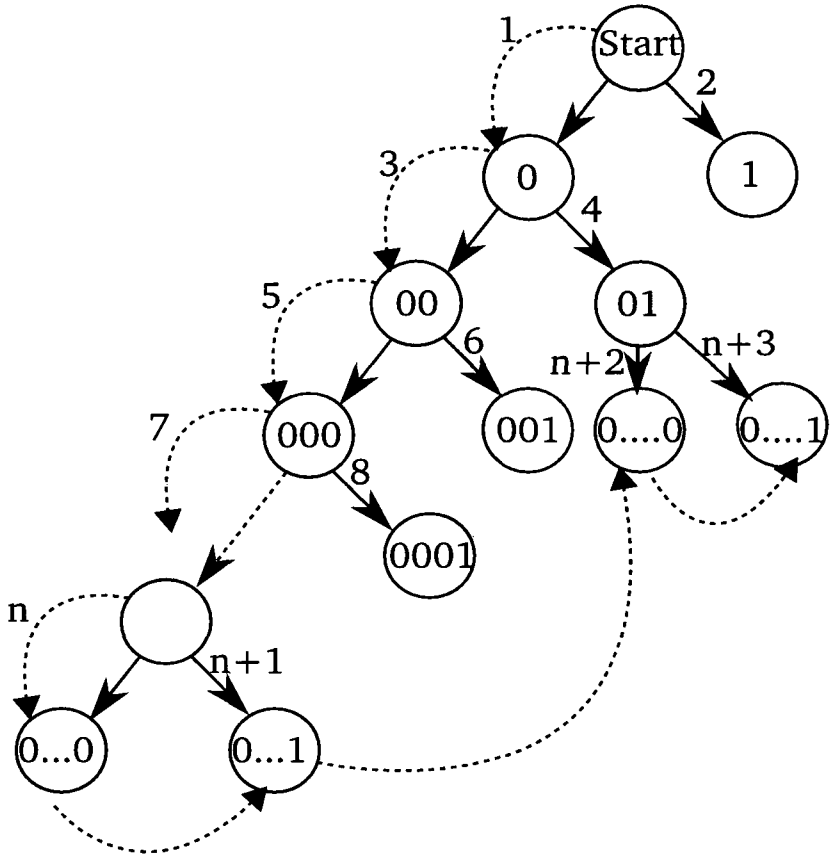


Figure 6.2: RVC VLD binary searching tree.

MPEG-4 specification Table B.6 of Annex B [78] is listed in Table 6.1, which is the VLD table for mcbpc for I-VOPs and S-VOPs. In this table, the eight values refer to different chroma coded block pattern (cbp) of block 4 and 5. Table 6.2 is the generated VLD table by the proposed method. All the data with underscore in this table are media data, which means that this is not true decode value and the VLD table engine will keep search the next value when the underscore data is found. The VLD table engine will stop and report searching failure if "1" is found, which means an error code is detected. Otherwise, true decoded value from VLD table is returned and the decoding process for the syntax is completed.

6.4 Modeling Variable Length Decoding of MPEG-4 SP in CAL

CAL [79] is a dataflow and actor-oriented language, specified as a part of the Ptolemy II project at the UC Berkeley [80]. CAL language has concise syntax structure and is suitable

Table 6.1: VLC table of MPEG B-6.

code	mytpye	cbpc(56)
1	3	00
001	3	01
010	3	10
011	3	11
0001	3	00
0000 01	4	01
0000 10	4	10
0000 11	4	11
0000 0000 1	Stuffing	–

Table 6.2: Generated VLC table of MPEG B-6.

File1	File2	Output-File
1	3	
001	19	
010	35	Start index: 0
011	51	10, 12, 18,58,26
0001	4	76 34, 16, 42,50,1
0000 01	20	8, 144, 208, 140,
0000 10	36	204;
0000 11	52	

for specifying complex signal processing systems as MPEG decoders.

Figure 6.3 shows that the graphical representation of the CAL model of the MPEG-4 SP decoder [78]. The Open Dataflow environment [81] is used to design and simulate CAL models. The decoder includes several networks of actors. The incoming bit stream is at first converted into sequential bits by the "serialize" FU, and then is decoded by the "Parser", the part of RVC-CAL source code is listed in Appendix C.1. The "TextureDecoding" and "MotionCompensation" networks of actors contain all the coding tools necessary for decoding the video. Figure 6.4 illustrates the inside of the "parser" FU present in Fig. 6.3. It shows how VLD FUs are connected to the parser for decoding Variable Length codes. For the sake of clarity, Fig. 6.4 represents only the connection of one VLD FU to the parser. This VLD FU serves at decoding the DCT coefficients (Table B-16 of Annex B of the MPEG-4 standard [78]). The FU "parser" is generated automatically by the XSLT process (refer to Section. 6.5). The VLD FU is generated using the process described in Section. 6.3. The "BlockExpand" FU is part of the MPEG toolbox. It outputs the AC coefficients.

When the parser meets a Variable Length code, it consumes only one bit from the bit

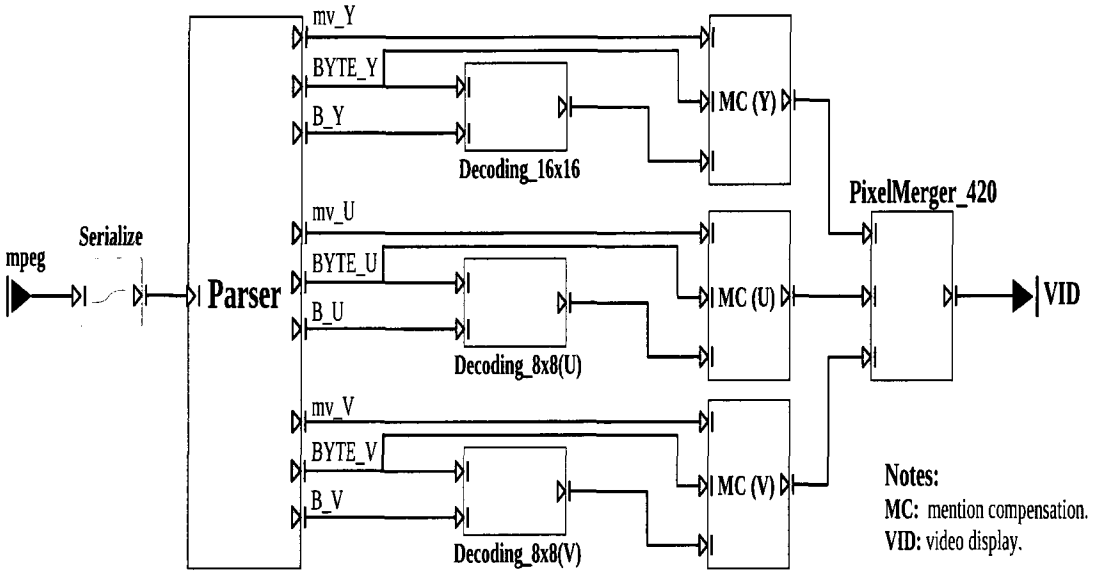


Figure 6.3: RVC CAL model of MPEG-4 simple profile.

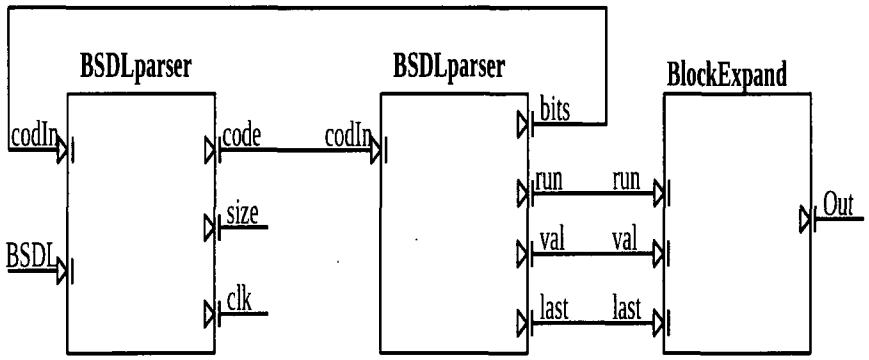


Figure 6.4: RVC VLD function unit.

stream port. It sends it to the VLD FU. If there is no entry in the table which corresponds to the input bit, the VLD FU sends back to the parser a token noticing that no matching has been found. Thus, the parser consumes an additional bit and sends it to the VLD FU. This latter will check if the first bit and the newly received bit match an entry in the table. If no, it continues sending token to the parser, saying that there is no matching and the parser must send an additional bit. If yes, the VLD FU sends a token to the parser saying that a matching has been found and the parser can parse the next element of the bit stream. The result of the parsing is then outputted by the VLD FU to the "BlockExpand" FU.

The source code of the VLD FU for decoding the "mcbpc" variable code is shown in

Appendix C.2. The only part of the FU which is automatically generated is a list of numbers, representing the VLD table. The rest of the code is always the same for all the VLD FUs. The extra code is needed to handle the optimized list of number representing the VLD table. This section showed how the Variable Length Decoding process has been modelled in CAL. The next section shows how the parser handles the communications with the VLD FUs to decode these variable length codes.

6.5 From Bit stream Scheme to Parser

Video coding is used under the various multimedia applications such as video conferencing, digital storage media, television broadcasting, and internet streaming. Due to the heterogeneity of modern networks and terminals, current multimedia technology has to deal with different user's requirements. As such, the use of scalable video coding, which derives useful video from subsets of a bit stream, is a must. RVC is compatible with SVC very well and it can implement SVC in function unit level. At this moment, the solution is that the MPEG-21 multimedia framework enables transparent and augmented use of multimedia resources across a wide range of networks and devices used by different communities [82].

The BSDL parser is a primordial functional unit in the RVC framework because it feeds the coding tool chain with the information contained in the bit stream to be decoded. As RVC is a framework for rapid development of decoding solution, the structure of the bit stream can be modified in order to explore the design space. To avoid the designer to write it by hand (which would be very time-consuming and error prone), a method has been developed to generate directly a parser from the bit stream syntax [76]. Figure 6.5 shows that the components of this transformation process. Each component is implemented in a separate XSLT style sheet.

Pre-processing is the first operation conducted by the top level style sheet. The pre-processing collects the individual schemata into a single intermediate tree, taking care to correctly manage the namespace of each component schema and also performs a number of other tasks, including assigning names to anonymous types and structures. Finite state machine (FSM) design is the major component of the parser actor. The FSM schedules the reading of bits from the input bit stream into the fields in the various output structures, along with all other components of the actor. The FSM is specified as a set of transitions, where each transi-

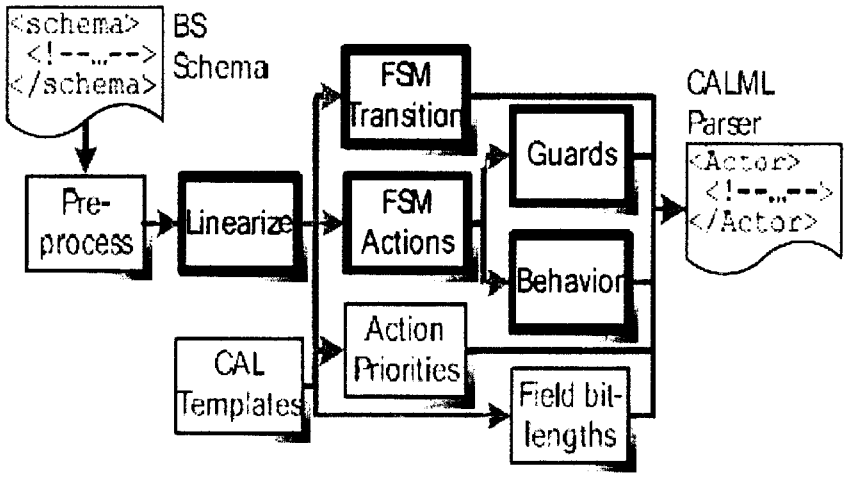


Figure 6.5: XSLT transformation process: BSDL to CAL.

tion has an initial state, a final state, and an action. BSDL specifies that the order of options within a choice establishes their priority: the first option has priority over the second, and so on. These priorities are recorded in the actor as priorities between the test actions. Guard expressions are built from the control flow constructs in the BSDL schema. The behaviour of each action is to complete such tasks as storing data in the appropriate location in the output structure. Finally, the CAL component declares templates for each of the constructs in the language, such as a FSM schedule, a function call, or an assignment. These templates are called by other components of the style sheet when building the actor. Collecting all of the CAL syntax into a single style sheet also means that an alternative style sheet could be provided in place of the CAL sheet. The source code shown in Appendix C.3 illustrates a part of the parser automatically generated from the bit stream schema. It shows the actions and the finite state machine generated for handling the communication between itself and external VLD FUs. When the parser meets a variable length code, the actions shown in Appendix C.3 are generated. First, the parser reads one bit from the bit stream input port (DCT_Coeff.read action). The next step consists in sending the bit to the corresponding VLD table; it is done in action DCT_Coeff.output. Then, the parser waits for a token coming from the VLD FU. This token (finish) indicates if a matching has been found in the table or not. If yes, the value of finish is true and the action DCT_Coeff.finish is fired and the number of bits to read for

the next element is set. If not, the value of finish is false and the DCT_Coeff.notFinished is fired and one more bit must be read (M4V_VLC_LENGTH = 1). The finite state machine summarizes the transitions. This section shows how the variable length decoding process is handled by the generated parser to decode variable length codes.

6.6 Hardware and Software Implementation

The important reason for which CAL has been adopted as language specifying the reference software of the RVC toolbox is that CAL is suitable for direct synthesis of "efficient" software and hardware by means of CAL2SW and CAL2HW tools [83, 84]. Furthermore, the very interesting aspect of this framework is that CAL models are used as inputs both for the hardware and software code generators. Thus software and hardware implementations can be derived from a unique CAL model. The designer develops a unique model and can generate seamlessly hardware and software implementation of CAL actors.

As the code of the VLD actors and parser are very simple, the generation of efficient code is straightforward. In [84], it has been shown that the hardware implementation of the MPEG-4 SP decoder modeled in CAL is more efficient than the one designed by hand in VHDL. Furthermore, in terms of coding effort, it took twice less time for a designer to write the CAL model than the VHDL model.

6.7 Conclusions

Reconfigurable video coding framework is introduced in this chapter. An efficient VLD toolbox can be automatically generated by the proposed design. It is successfully implemented in CAL and validated by simulations. This chapter shows that it is possible to dynamically generate a RVC parser using a BSDL description of the bit stream and assembling RVC decoding FUs from the standard RVC toolbox.

Chapter 7

Reconfigurable Video Coding – DV/DVCPRO

7.1 Introduction

Video coding solutions based on a pre-defined video coding standard have certain limitations when new standards are being added. How to utilize their commonalities and reduce design time is of great concern. Writing reference code for a new standard starts with scratch, which is time consuming and labor intensive. On the other hand, sequential C/C++ code aiming at functional validation hides intrinsic concurrency and parallelism. Consequently, converting the sequential code into pipeline and multicore process requires architecture re-building and re-writing code. In other words, the complex C/C++ specifications no longer constitute a good starting point for implementing the standards. It is preferred to develop a framework to operate at a higher level of abstraction and simplify top-down system development and design. To deal with this issue, MPEG organization launched a new standard called reconfigurable video coding (RVC) in 2006 [85]. Some video coding standards have been successfully implemented with RVC framework [76, 86, 87]. In 2007, the work in [76] initiatively implemented the MPEG-4 simple profile with the RVC framework. The work in [86] reported implementation of AVS intra decoder with RVC framework. The work in [87] recently reported that an efficient H.264/AVC baseline encoder had been built by the RVC-CAL dataflow components.

In this chapter, we implement the other video coding standard, DV/DVCPRO, using the

RVC framework. The remaining of the chapter is organized as follows: the MPEG RVC framework and DV/DVCPRO standard are introduced in Section 7.2 and Section 7.3. The proposed design with RVC framework is presented in Section 7.4. Section 7.5 provides the experimental results and analysis. Section 7.6 concludes this chapter.

7.2 Reconfigurable Video Coding

The objective of RVC framework is to describe current and future codecs in a way that makes commonality explicit and reduces the implementation burden for device vendors [76]. The key difference between RVC and conventional codec standards is their conformity point. The conventional codec standards define their conformity point at decoder level whereas MPEG RVC defines it at tool level so that MPEG RVC exhibits much more flexibility. Hence several configurations of components, taken by previous monolithic specifications, are possible [6]. MPEG RVC framework is initiatively motivated by the following observations:

1. Supporting multiple standards: Video coding standards have been changed for decades. New multimedia devices or development platforms need to support multiple codecs, such as MPEG-1, MPEG-2, MPEG-4, H.264/AVC and DV/DVCPRO.

2. Interoperability: Commonality and similarity between the standards have not been utilized efficiently. It is urgent to develop a new standard to incorporate the commonality and similarity in order to reduce design time.

3. Obstacles of current specification:

   - The normative specifications (written in generic C/C++) do not expose the potential parallelism which is intrinsic to the algorithms constituted in the codecs. They are excessively large and hard to read.
   - The reference code written in complex sequential C/C++ is labor intensive to transform to Verilog or the new generation codes of multicore platform stream processor.

Thus, the goal of the MPEG RVC standard is to offer a high-level algorithm model to innovate MPEG standards in a way that is competitive in current dynamic environment, thereby

enabling MPEG to continue serving the needs of the industry in terms of video coding standards. An additional challenge taken by MPEG RVC is to provide a easy-going implementation model for efficient hardware and software synthesis. The following three components are mainly included in RVC framework.

- Video coding tools library (VTL): The normative library is specified by textual specification and corresponding reference software, written with RVC-CAL language [79] to specify each library component.
- FU network language (FNL): The normative language is XML dialect. It specifies decoder configuration, and interconnected network and parameterization of standard library components.
- RVC bit stream description language (RVC-BSDL): The normative language describes the syntax of a new configuration of a MPEG RVC decoder.

Unlike other video coding standards, MPEG RVC decodes the configuration information before the video bit stream. To decode a video bit stream, the decoder needs to know: (a). How to parse the bit stream; (b). How to decode these elements. The MPEG RVC decoding engine receives RVC-BSDL and FNL specifications in compressed form. The decoder composition module generates a decoding solution (an actual video decoder) based on the RVC-BSDL and FNL specifications. It makes use of selected function units (FUs) from VTL and connects them according to FNL specification. Once the decoding solution has been generated, it can then decode the bit stream. This approach has a number of potential benefits. A decoder can be modified to decode a different format by sending new RVC-BSDL/FNL descriptions and enabling efficient support for multiple coding formats. Moreover, non-standard coding format can be supported provided it uses FUs available to the decoder (i.e. FUs in the decoder's VTL).

7.3 DV/DVCPRO Standard

Digital video (DV) is a digital video format created by a group of companies [1], and launched in 1995 [88, 89]. DV refers to the compression format employed to capture, edit and store

[1]Led by Sony, JVC, Panasonic and other producers

video footage. Moreover, DV is also applied to cameras that record using mini-DV tape.

Two standards [88, 89] define the data structure for the interface of DV-based digital audio, subcode data, and compressed video at different bit rates: the DV standard includes both 525/60 and 625/50 systems, in which the numeric values "525" and "625" refer to the number of the horizontal sync lines while the numeric values "60" and "50" indicate the field rate; The DVCPRO standard now includes 1080/60i, 1080/50i, 720/60p, and 720/50p systems, in which the numeric values "1080" and "720" refer to "1920 × 1080" and "1280 × 720" image sampling structure, respectively, while the values "50" and "60" refer to the field/frame rate and the letters "i" and "p" indicate the field/frame type. There are three types of compressed bit rates defined in DV/DVCPRO standard: 25Mbps, 50Mbps, and 100Mbps, as shown in Table 7.1. This expansion allows the DVCPRO format to support not only a high quality program production system but also the next generation of broadcast system. A broadcast system based on DVCPRO's 25Mbps, 50Mbps and 100Mbps compression is the most efficient, practical, and widely supported method of providing today's requirements.

Table 7.1: Formats of DV standards.

DV-Standards (Types)	Size	Format	Channels	Sequences	DIF-Size (bytes)	System	DataRate (Mb/s)	Specification
DV-NTSC	720x480	4:1:1	1	10	120000	60Hz	25	IEC61834
DV-PAL	720x576	4:2:0	1	12	144000	50Hz	25	IEC61834
DV25-NTSC	720x480	4:1:1	1	10	120000	60Hz	25	SMPTE314
DV25-PAL	720x576	4:1:1	1	12	144000	50Hz	25	SMPTE314
DVCPRO50-NTSC	720x480	4:2:2	2	10	240000	60Hz	50	SMPTE314
DVCPRO50-PAL	720x576	4:2:2	2	12	288000	50Hz	50	SMPTE314
DVCPRO100P720P-NTSC	960x720	4:2:2	2	10	240000	60Hz	100	SMPTE370
DVCPRO100P720P-PAL	960x720	4:2:2	2	12	288000	50Hz	100	SMPTE370
DVCPRO100HD-NTSC	1280x1080	4:2:2	4	10	480000	60Hz	100	SMPTE370
DVCPRO100HD-PAL	1280x1080	4:2:2	4	12	576000	50Hz	100	SMPTE370

The number of channels of the compressed DV stream is assigned to 1, 2 and 4 corresponding to DV/DV25, DVCPRO50, and DVCPRO100 formats, respectively [88, 89]. Each channel is further divided into 10 sequences for 525/60 system and 12 sequences for 625/50 system. Each sequence consists of header, subcode, Video Auxiliary information (VAUX), audio, and video sections as shown in Fig. 7.1. Each section is comprised of numbers of digital interface (DIF) blocks. The DIF block is the basic element of DV data structure and each DIF block consists of a 3-byte ID and 77 bytes of data. DIF data bytes are numbered 0 to 79. The type of DIF block is assigned by its ID and the DIF data part (play load) presents the parameters of the DV decoder.

The DV/DVCPRO standard only has intra (I) frame and its process is straightforward.

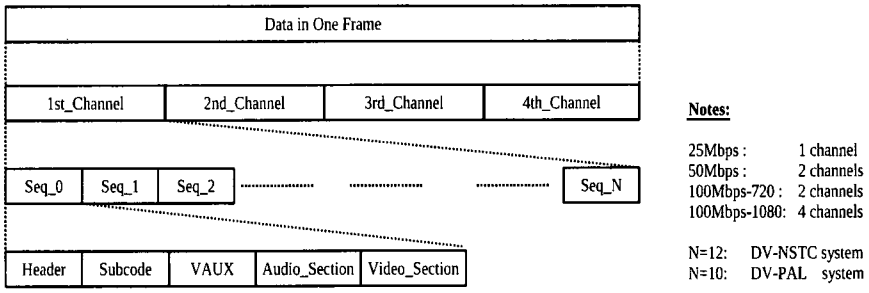


Figure 7.1: DV data type.

Figure 7.2 illustrates the procedure of the complete process. The decoder generates the video output by the following processes: variable length decoder (VLD), inverse quantization (IQ), weighting, inverse discrete cosine transform (IDCT), block de-shuffling and upsampling while the audio output is generated by data mapping and data de-shuffling units. One of the most important differences between DV/DVCPRO and MPEG compression is that the audio and video data of DV are mixed into DIF blocks. In the 1080-line system, video data, audio data, and subcode data in one video frame are processed in each frame. In the 720-line system, these data are spread into two video frames. To process the 720-line system in the same way as the 1080-line system. They are processed within one frame duration of the 1080-line system. The audio data, corresponding to one video frame in the 1080-line system and two video frames in the 720-line system, is defined as an audio processing unit [89].

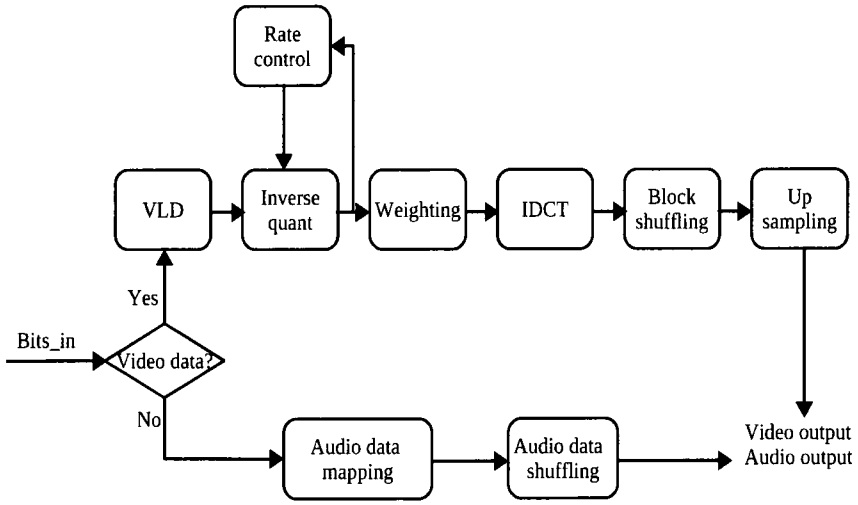


Figure 7.2: DV decoder data processing block diagram.

7.4 Implementation & Design

As stated above, DV/DVCPRO is intra frame only video standard without bidirectional (B) and progressive (P) frames so that its operation is not as complex compared to other standards, such as MPEG-4 and H.264. However, there are challenges when implementing DV/DVCPRO with RVC-CAL for the following reasons: How to efficiently partitioning the FUs while considering the following features of DV data: 1). Video and audio data are shuffled in DV compressed data; 2). There are more than nine types of video formats in DV/DVCPRO standard; 3). There are various processing modules, such as 8-8 inverse discrete cosine transform (IDCT), 2-4-8 IDCT, scanning, weighting, de-shuffling, etc. At first, the partition of DV FUs are important for RVC-CAL implementation. Efficient FU partition, utilizing available FUs, is able to save design time and improve performance. Based on the features of DV/DVCPRO processing, our proposed RVC implementation mainly contains three parts as shown in Fig. 7.3: Parser FUs, VLD & IDCT FUs, and De-shuffling FUs. This partition divides DV/DVCPRO into reasonable function blocks in order to minimize the number of tokens between FUs and actors while considering the reuse of available MPEG-RVC FUs. For example, 8-8 IDCT and 2-4-8 IDCT are separated in order to use the available MPEG-4 8x8 IDCT FU. The audio process is separated from other FUs since no reference FUs are available.

7.4.1 RVC Parser FUs

The parser FUs unit tries to decode DV parameters for the following using. Unlike other video data, DV data has strict DIF with a size of 80 bits and data location, which makes the design simpler than others. The RVC-CAL parser unit consists of ten interacting actors. At first, the "Serial" actor reads the input DV data and makes them in the form of tokens. The following actors consume the incoming tokens according to different ID type: Header (ID=000), Subcode (ID=001), VAUX (ID=010), and AAUX (ID=011). When these actors consume the incoming tokens, the parsing process is considered complete. "Header" actor produces the DIF sequence number, DIF block number, and channel identification tokens while "Subcode" actor generates time code (TC) and binary group (BG) package tokens. The actors "VAUX" and "AAUX" are further separated into "VS, VSC, AS" and "ASC" actors according to the different incoming

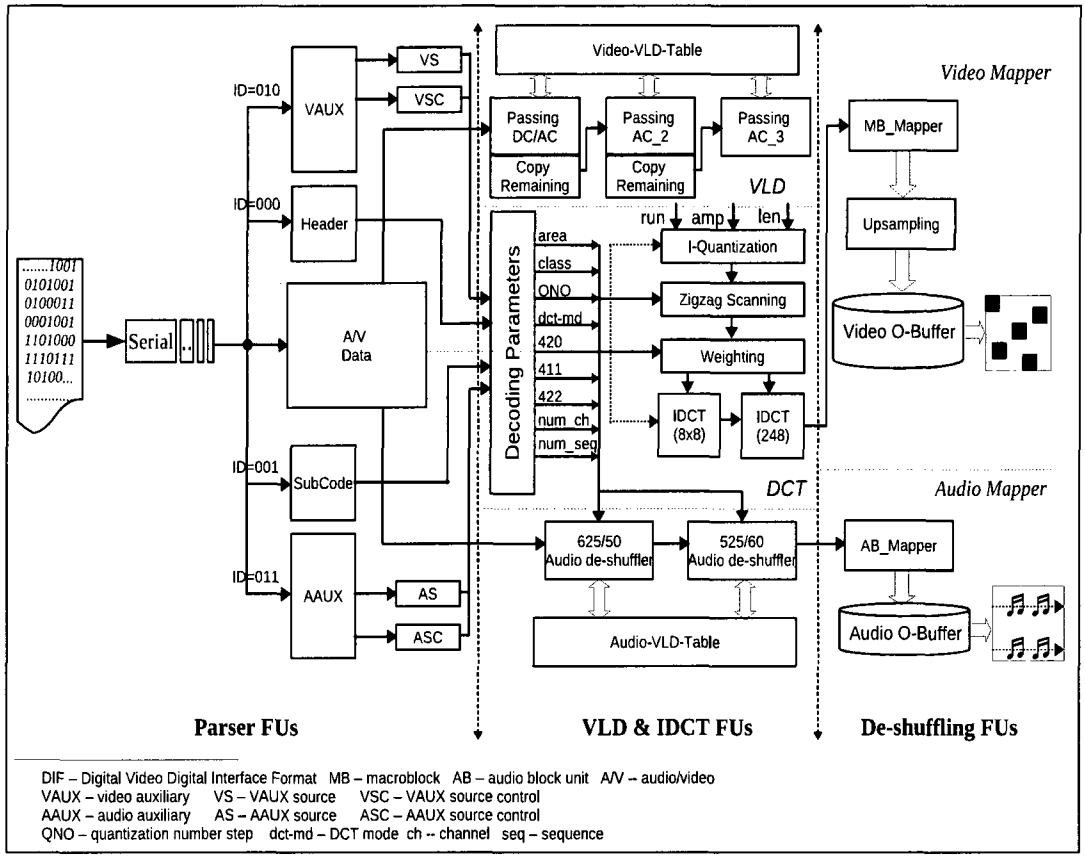


Figure 7.3: DV-FU partition & implementation.

tokens. The frame type, display type and decoded type are parsed by "VSC" and "VS" actors while audio compressed mode and sampling frequency are parsed by "ASC" and "AS" actors. The actual compressed video and audio data tokens are produced by the "A/V data" actor.

7.4.2 RVC VLD & IDCT FUs

The variable length decoder (VLD) actor of DV standard is different from MPEG standard. As shown in Fig. 7.3, three passes have been adopted for the DV VLD. The procedure of VLD is: First, passing the bit stream data into the first pass; Second, passing the remaining data into the second pass if there are surplus data remained in the current DIF block; Third, passing the remaining data into the third pass if there are surplus data remained in current MB; Finally, the VLD process is terminated whether there are data remained or not. The VLD table is automatically generated using similar procedure in [7].

The IDCTs of DV standard have two modes: 8-8-IDCT and 2-4-8-IDCT. They are selectively used to optimize the data-reduction process, depending upon the degree of content variations between the two fields of a video frame. Based on 8-8 DCT and 2-4-8 DCT modes, there are two modes of scanning and weighting, accordingly. The 8-8 scanning IDCT mode is kind of zigzag scanning, which is similar to MPEG-4, hence, this allows one to make little modification and reuse the FU from MPEG-4. The 2-4-8 mode needs further modifications from MPEG-4. In this mode, one 8x8 macroblock (MB) is divided into two vertical 8x4 sub blocks so that the scanning order has to be modified accordingly. Therefore, the left top pixel of the upper sub block is scanned at first and followed by the left top pixel of the lower sub block. The DCT coefficients are weighted by a quantizer matrix. The different quantizer matrices are set for different luminance and color signals [88, 89]. Also, different DV formats have different quantizer matrices and therefore different weighting values. DV DCT coefficients are quantized to within 9-bit words in order to limit the amount of data in one video segment to five compressed MBs.

7.4.3 RVC De-shuffling FUs

The RVC De-shuffling FUs contain video and audio data re-arrangement actors. "MB_Mapper" actor designates the correspondence between video DIF blocks and compressed macro blocks. "Video_De-shuffling" actor defines the correspondence between compressed macro blocks and the video segment. The video segment consists of five MBs which are assembled from various areas within the video frame. Note that the NSTC system follows different MB mapping and video shuffling rules.

The "AB_Mapper" actor outputs the decoded audio data according to the DV audio block mapping rule which is defined in DV/DVCPRO standard [89]. Note that NSTC and PAL system have different audio mapping rules to be followed.

7.5 Simulation & Analysis

The DV/DVCPRO decoder is successfully implemented in RVC-CAL simulation environment. CAL is not only a description formalism but is supported by a simulation environment portable on all platform supporting JAVA vitual machine (VM). This interpreter is currently integrated

into two modelling and simulation environments: Ptolemy II [80] and Moses [90].

Numbers of DV formatted sequences are tested by the proposed design, such as Foreman, Mobile and Calendar, and Driver and Flower. To verify the DV audio decoding, the tested video sequences are combined with raw audio data. The encoded DV format sequences can be generated by FFMPEG [91] reference code. We compare the results from the proposed design with RVC-CAL and ones from FFMPEG reference decoder both in video and in audio output. No differences are found. Both the decoded video and audio output are able to be played back successfully. Therefore, the feasibility of DV implementation with RVC-CAL is verified. The experimental analysis is made in the following aspects:

7.5.1 Reusability of MPEG-4 FUs

One of the major advantages of RVC framework is to re-use available FUs and reduce design time. As stated above, MPEG-4 simple profile has been successfully developed by MPEG organization. Some FUs are available to be used or can be modified for reuse. Unlike H.264/AVC or AVS standard, the DV/DVCPRO standard has less similarity with MPEG. Therefore, Some new FUs have to be re-designed, however, some MPEG-4 FUs can still be modified to be used in order to save design time. As shown in Fig. 7.3, modified IDCTs from MPEG-4 simple profile have been used in the proposed design while the inverse quantization FU is much similar as MPEG-4. We just reuse it as normal. The statistic reusability table is listed on Table 7.2.

Table 7.2: FUs reusability of DV.

No.	FunctionBlock	FU	Reused	New	Modified	Comments
1	Parser	ParserHeader		✓		Header,Subcode,VAUX,AAUX
2		Passing		✓		3 Passing for DC & AC,Remains
3		I-Quant	✓			Reuse MPEG-4
4	Decode_Video	Scaning			✓	Zagzig scanning for 8x8 and 2-4-8
5		Weighting			✓	Weighting for 8x8 and 2-4-8 mode
6		IDCT			✓	IDCT for 8x8 and 2-4-8 mode
7	Video_De-shuffling	Video_Mapping		✓		Mapping DIFs with MBs
8		UpSampling		✓		Upsample420 to 422, or 411 to 422
9		Audio_PAL_De-shuffling		✓		De-shuffling PAL Audio DIFs
10	Decode_Audio	Audio_NTSC_De-shuffling		✓		De-shuffling NTSC Audio DIFs
11		AB_Mapping		✓		Mapping Audio Blocks for Output

7.5.2 Reduction of Design Overhead

Another advantage of RVC framework is that it has high abstract and concise code representation. Table 7.3 shows the lines of code (LOC) compared with reference code reported in [91].

The numbers show that the RVC-CAL implementation has an average 25.8% less LOCs than the reference C code from FFMPEG. We can not compare the development time in detail because it is hard to know how long writing the reference code has been taken. However, writing the RVC-CAL code only takes 3 months for a middle-level CAL programmer, which is more efficient than writing C code.

Table 7.3: Comparison between C code and RVC-CAL.

Name	C code	RVC-CAL	Reduction (%)
Video Process	3718	2927	21.3
parser	–	978	–
VLD	–	365	–
IQ	–	202	–
Scanning	–	166	–
Weighting	–	178	–
IDCT-8x8	–	229	–
IDCT-248	–	281	–
Video de-shuffler	–	144	–
Upsampling	–	256	–
Video mapper	–	128	–
Audio Process	835	451	46
Audio de-shuffler	–	238	–
Audio block mapper	–	213	–
Total	4553	3378	25.8

7.5.3 Efficient Code Transformer

The last advantage of RVC framework is that it automatically targets both software and hardware with supporting tools. An automatic CAL-to-C code generator has been developed in [83] while CAL-to-VHDL code generator is successfully developed in [92]. It is reported that the automatically generated VHDL is not only four times faster in development time, but it is also more efficient in execution time. The main reason is attributed to RVC-CAL being using dataflow methodology instead of direct VHDL register transfer level (RTL) design.

7.6 Conclusions

In this chapter, a RVC-CAL implementation of DV/DVCPRO video coding standard has been featured and hence the validation of RVC implementation has been proved. Based on the fea-

tures of DV/DVCPRO standard and RVC framework, RVC FUs partition has been described. The functions of important actors and tokens are also explained in detail. The experimental result illustrates the advantages of using MPEG RVC as a new video coding standard. Writing RVC-CAL takes less time than writing reference C code or VHDL code, which allows developers to concentrate on function optimization rather than coding skills. Moreover, RVC framework enhances the interoperation between standards. Not only can available function units be reused, but new function units and algorithms, or even rebuilt standards, can also be incorporated into the RVC framework. Finally, RVC-CAL adopts the parallelism and dataflow programming methodology, which is closer to hardware implementation than sequential process.

Chapter 8

Concluding Remarks

8.1 Conclusions

Algorithms and implementation are the major aspects of video coding standards, particularly when new technologies and standards are gradually emerging. This thesis started with low-complexity and efficient H.264/AVC algorithms, MPEG-2/H.264 transcoding algorithms and ended with a reconfigurable video coding framework implementation. The aim of the thesis is to provide solutions for three major problems the modern video coding standards are facing: Fast and efficient algorithms of H.264/AVC; Transition from one standard to the other one; Utilizing commonalities between standards and reducing repeating design. The major contributions are listed as follows:

1. Low-complexity H.264/AVC algorithms:

   (a) Intra 4x4 prediction: Three techniques have been contributed in this thesis: a fast parallel architecture, redundancy reduction algorithm and complexity-reduced mode decision algorithm. A significant reduction in execution time is achieved without significant loss of video quality. Only 204 cycles are required to process a macroblock (MB). Compared to the dedicated intra prediction [28], processing speed is enhanced by 79%.

   (b) Deviation based rate control algorithm: Three contributions have been made in this thesis: (1) The multiple quantization parameters (QPs) determination algorithm is based on the statics of the deviation measure. Thus, relatively accurate QP

estimation can be achieved because it takes the features of the current frame into consideration. (2) Slice-based rate control scheme stabilizes the output bit rates. (3) The scene change is detected with the complexity of adjacent frames based on deviation measure. The proposed algorithm not only generates stable output bit rate but also improves the video quality. Compared to the JM12.0 under various sequences, the proposed algorithm improves the average PSNR by 0.76dB while keep the mismatch of the bit rate output less than 8%.

2. MPEG-2/H.264 transcoder:

   (a) This research work contributes three efficient mapping algorithms: motion vector (MV) mapping; block size mapping and frame/field mapping;

   (b) This research work also contributes an efficient ranking-based, rate-distortion optimized mode decision algorithm;

   (c) These algorithms achieve good rate-distortion performance with low complexity. Compared to cascaded decoder-encoder solution, the R-D performance is maintained while the complexity is significantly reduced.

3. RVC framework:

   (a) This research work contributes an efficient data flow based implementation of the variable length decoding (VLD) process, particularly adapted for the instantiation and synthesis of CAL parsers in the MPEG RVC framework.

   (b) DV/DVCPRO has been examined and modelled with MPEG RVC framework. The flexibility and ease of RVC-CAL is demonstrated as well as the validation of RVC implementation.

8.2 Future Works

Even though the thesis provided methods to solve the three major issues mentioned above, there are some algorithms and implementations to be further developed:

1. H.264/AVC efficient algorithm: Applying the available deviation algorithm to "P" and "B" frames;

2. Transcoding: With the transcoding algorithms presented in this thesis, the current transcoder's complexity is dominated by sub-pel interpolation and motion refinement. And mode decision still has considerable complexity. Therefore, it will be interesting to further explore along these directions to have more efficient transcoder design without affecting the compression efficiency.

3. Reconfigurable video coding: Implementing H.264/AVC decoder with reconfigurable video coding framework.

Bibliography

[1] J. Li and E. Abdel-Raheem, "Fast Implementation of H.264 4x4 Intra Prediction," *IEICE Electronics Express*, vol. 7, no. 5, pp. 332–338, Mar. 2010.

[2] ——, "Efficient Rate Control for H.264/AVC Intra Frame," *Submitted to IEEE Transactions on Consumer Electronics*, Apr. 2010.

[3] J. Xin, J. Li, A. Vetro, and S. ichi Sekiguchi, "Motion Mapping and Mode Decision for MPEG-2 to H.264/AVC Transcoding," *Multimedia Tools Application*, vol. 35, no. 2, pp. 203–223, May 2007.

[4] J. Xin, J. Li, A. Vetro, H. Sun, and S. ichi Sekiguchi, "Motion Mapping for MPEG-2 to H.264/AVC Transcoding," *IEEE International Symposium on Circuits and Systems*, pp. 1991–1994, May 2007.

[5] J. Xin, J. Li, A. Vetro, and H. Sun, "Motion Mapping for MPEG-2 to H.264/AVC Transcoding," *Mitsubishi Electric Research Laboratories TR2007-085*, Apr. 2008.

[6] J. Li, D. Ding, C. Lucarz, S. Keller, and M. Mattavelli, "Efficient data flow variable length decoding implementation for the MPEG reconfigurable video coding framework," *IEEE Workshop on Signal Processing Systems*, pp. 188–193, Oct. 2008.

[7] C. Lucarz, J. Li, D. Ding and M. Mattavelli, "Automatic-generation of RVC Parser from BSDL Syntax Description: Variable Length Decoding," *ISO/IEC JTC1/SC29/WG11 MPEG/M15163, Antalya, Turkey*, Jan. 2008.

[8] C. Lucarz, D. Ding, J. Li and M. Mattavelli, "BSDL Description of MPEG-4 SP

and AVC BP Bitstream Syntax for RVC Framework," *ISO/IEC JTC1/SC29/WG11 MPEG/M15159 Jan.2008, Antalya, Turkey*, Jan. 2008.

[9] C. Lucarz, J. Li, D. Ding and M. Mattavelli, "Function Units for RVC Toolbox: Variable Length Decoding," *ISO/IEC JTC1/SC29/WG11 MPEG/M15164 Jan.2008, Antalya, Turkey*, Jan. 2008.

[10] J. Li and E. Abdel-Raheem, "Modeling DV/DVCPRO Standards on Reconfigurable Video Coding Framework," *Accepted by Journal of Electrical and Computer Engineering, Hindawi Publishing Corporation*, Apr. 2010.

[11] Iain E. G. Richardson, *H.264 and MPEG-4 Video Compression Video Coding for Next-generation Multimedia.* Wiley, 2003.

[12] Y. Wang, J. Ostermann, and Y.Q. Zhang, *Video Processing and Communications.* Prentice Hall, 2002.

[13] L. Hanzo, P.J. Cherriman and J. Streit, *Video Compression and Communications: H.261, H.263, H.264, MPEG4 and HSDPA-Style Adaptive Turbo-Transceivers.* Wiley, 2007.

[14] K. R. Rao, Z. S. Bojkovic and D. A. Milovanovic, *Multimedia Communication Systems: Techniques, Standards and Networks.* Pearson Education, 2002.

[15] K. R. Rao and J. J. Hwang, *Techniques and Standards for Images, Video and Audio Coding.* Prentice Hall, 1996.

[16] ISO/IEC 11172-2, "Information Technology – Coding of Moving Pictures and Associated Audio for Digital Storage Media at Up to About $1.5M/bits$ - Part 2: Video," 2003.

[17] CCITT SG XV, Draft Revision of Recommendation CCITT H.261, "Video Codec for Audiovisual Services at P $\times$ 64 kbits/s," Mar. 1990.

[18] ISO/IEC 13818-2, "Information Technology – Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video," 2000.

[19] ITU-T Rec. H.263, "Video Coding for Low Bit Rate Communication," vol. 3, Nov. 1995.

[20] ISO/IEC 14496-2, "Information Technology – Coding of Audio-Visual Objects – Part 2: Visual," 2004.

[21] T. Wiegand, G. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, Mar. 2003.

[22] Z. Zhou, J. Xin and M.T. Sun, "Fast Motion Estimation and Inter-mode Decision for H.264/MPEG-4 AVC Encoding," *Journal of Visual Communication and Image Representation*, vol. 17, no. 2, pp. 243–263, Apr. 2006.

[23] C. Kim, H. Shih, and C. Kuo, "Fast H.264 Intra-prediction Mode Selection Using Joint Spatial and Transform Domain Features," *Journal of Visual Communication and Image Representation*, vol. 17, no. 2, pp. 290–310, Apr. 2006.

[24] F. C. Pereira and T. Ebrahimi, *The MPEG-4 Book*. Prentice Hall, 2002.

[25] ETSI TS 101 154 V1.9.1, "Digital Video Broadcasting (DVB): Specification for the Use of Video and Audio Coding in Broadcasting Applications Based on the MPEG-2 Transport Stream," Sep. 2009.

[26] ISO/IEC 15938-4, "MPEG-7 Final Draft International Standard (FDIS)– Part 4," 2001.

[27] ISO/IEC JTC1/SC29/WG11 N4041, MPEG Requirements Group, "MPEG-21 Overview," *Singapore MPEG Meeting*, Mar. 2001.

[28] Y.-W. Huang, B.-Y. Hsieh, T.-C. Chen, and L. Chen, "Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder," *IEEE Trans. Circuit and Systems for Video Technology*, vol. 15, no. 3, pp. 378–401, Mar. 2005.

[29] W. Lee, S. Lee, and J. Kim, "Pipelined Intra Prediction Using Shuffled Encoding Order for H.264/AVC," *TENCON 2006*, pp. 14–17, Nov. 2006.

[30] G. Jin and H.-J. Lee, "A Parallel and Pipelined Execution of H.264/AVC Intra Prediction," *IEEE International Conference on Computer and Information Technology, CIT'06,*, pp. 246–250, Sep. 2006.

[31] K. Suh, S. Park and H. Cho, "An Efficient Hardware Architecture of Intra Prediction and TQ/IQIT Module for H.264 Encoder," *ETRI Journal*, vol. 27, no. 5, pp. 511–524, Oct. 2005.

[32] G. Sullivan, P. Topiwala, , and A. Luthra, "Fast 4x4 Intra-prediction Based on The Most Probable Mode in H.264/AVC," *IEICE Electronics Express*, vol. 5, no. 19, pp. 782–788, Oct. 2008.

[33] G. Tian, T. Zhang, X. Wei, T. Ikenaga, and S. Goto, "An Efficient Fast Mode Decision Algorithm for H.264/AVC Intra Prediction," *Image and Signal Processing of CISP Congress, Sanya, Hainan*, vol. 1, pp. 411–415, May 2008.

[34] F. Pan, X. Lin, S. Rahardja, K. P. Lim, Z. G. Li, D. Wu, and S. Wu, "A Novel Hardware Architecture of Intra-Predictor Generator for H.264/AVC Codec," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 7, pp. 813–822, Jul. 2005.

[35] S. Kwak, J. Kim, and D. Har, "Fast Mode Decision Algorithm for Intraprediction in H.264/AVC Video Coding," *IEICE TRANS. INF. & SYST.*, vol. E91-D, no. 7, pp. 2083–2086, Jul. 2008.

[36] C. sung Kim, Q. Li, and C.-C. J. Kuo, "Fast Intra-Prediction Model Selection for H.264 Codec," *SPIE International Symposium ITCOM 2003*, vol. 5241, pp. 99–110, Nov. 2003.

[37] G. Sullivan, P. Topiwala, , and A. Luthra, "The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions," *SPIE Conference on Applications of Digital Image Processing*, Aug. 2004.

[38] Joint Model Reference Software Version 10.2. [Online]. Available: http://iphome.hhi.de/suehring/tml/

[39] J. Li and M. Ahmadi, "Realizing High Throughput Transforms of H.264/AVC," *IEEE International Symposium on Circuits and Systems*, pp. 840–843, May 2008.

[40] R. Kordasiewicz and S. Shirani, "On Hardware Implementations Of DCT and Quantization Blocks for H.264/AVC," *The Journal of VLSI Signal Processing on Springer*, vol. 47, no. 2, pp. 93–102, May 2007.

[41] J. Xin, A. Vetro, and H. Sun, "Converting dct coefficients to h.264/avc transform coefficients," pp. 939–946, Jun. 2004.

[42] ISO/IEC 14496-10, "Information Technology – Coding of Audio-Visual Objects – Part 10: Advanced Video Coding," 2004.

[43] N. Kamaci, Y. Altunbasak, and R. M. Mersereau, "Frame Bit Allocation for H.264/AVC Video Coder Via Cauchy-Density-Based Rate and Distortion Models," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 8, pp. 994–1006, Aug. 2005.

[44] S. W. Ma, W. Gao, F. Wu, and Y. Lu, "Rate Control for JVT Video Coding Scheme with HRD Considerations," *Proceedings of the IEEE International Conference on Image Processing, Barcelona, Spain*, pp. 793–796, Sep. 2003.

[45] Z. G. Li, F. Pan, K. P. Lim, G. N. Feng, X. Lin, and S. Rahardaj, "Adaptive Basic Unit Layer Rate Control for JVT, Joint Video Team," *ISO/IEC JTC1/SC29/WG11 and ITU-T*, May 2003.

[46] H. Wang and S. Kwong, "Rate-Distortion Optimization of Rate Control for H.264 With Adaptive Initial Quantization Parameter Determination," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 1, pp. 140–144, Jan. 2008.

[47] X. Jing, L. Chau, and W.-C. Siu, "Frame Complexity Based Rate-Quantization Model for H.264/AVC Intraframe Rate Control," *IEEE Signal Processing Letters*, vol. 15, no. 1, pp. 373–376, Sep. 2008.

[48] A. Armstrong, S. Beesley, and C. Grecos, "Selection of Initial Quantization Parameter for Rate Controlled H.264 Video Coding," *Research in Microelectronics and Electronics*, pp. 249–252, Jun. 2006.

[49] S. Zhou, J. Li, J. Fei, and Y. Zhang, "Improvement on Rate-distortion Performance of H.264 Rate Control in Low Bit Rate," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 8, pp. 996–1006, Mar. 2007.

[50] "Joint Model Reference Software Version 12.0." [Online]. Available: http://iphome.hhi.de/suehring/tml/

[51] S. W. Ma, W. Gao, Y. Lu, and H. Q. Lu, "Proposed Draft Description of Rate Control on JVT Standard Joint Video Team," *ISO/IEC JTC1/SC29/WG11 and ITU-T SG16/Q.6 Doc*, Dec. 2002.

[52] "ISO/MPEG-2 Test Model 5," Apr. 1993. [Online]. Available: http://www.mpeg.org/MPEG/video/mssg-free-mpeg-software.html/

[53] M. Jiang and N. Ling, "Low-delay Rate Control for Real-time H.264/AVC Video Coding," *IEEE Transactions on Multimedia*, vol. 8, no. 3, pp. 467–477, 2006.

[54] S.-C. Lim, H.-R. Na, and Y.-L. Lee, "Rate Control Based on Linear Regression for H.264/MPEG-4 AVC," *Signal Processing: Image Communication*, vol. 22, no. 1, pp. 39–58, 2007.

[55] Y. K. Tu, J.F. Yang and M.T. Sun, "Rate-Distortion Modeling for Efficient H.264/AVC Encoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 5, pp. 530–543, 2007.

[56] W. J. Kim, J. W. Yi, and S.-D. Kim, "A Bit Allocation Method Based on Picture Activity for Still Image Coding," *IEEE Transactions on Image Processing*, vol. 8, no. 7, pp. 974–977, Feb. 1999.

[57] S.-C. Hsia and S.-H. Wang, "Adaptive Video Coding Control for Real-time H.264/AVC Encoder," *Journal of Visual Communication and Image Representation*, vol. 20, no. 7, pp. 463–477, Jun. 2009.

[58] A. Vetro, C. Christopoulos and H. Sun, "Video Transcoding Architectures and Techniques: An Overview," *IEEE Signal Processing Magazine*, vol. 20, no. 2, pp. 18–29, Mar. 2003.

[59] J. Xin, C.W. Lin and M.T. Sun, "Digital Video Transcoding," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 84–97, Jan. 2005.

[60] H. Kalva and B. Petljanski, "Exploiting The Directional Features in MPEG-2 for H.264 Intra Transcoding," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 2, pp. 706–711, May 2006.

[61] L. Wang, Q. Wang, Y. Liu, and W. Lu, "A Fast Intra Mode Decision Algorithm for MPEG-2 to H.264 Video Transcoding," *IEEE 10th International Symposium on Consumer Electronics (ISCE)*, vol. 28, no. 1, pp. 1–5, Jun. 2006.

[62] B. Petljanski and H. Kalva, "DCT Domain Intra MB Mode Decision for MPEG-2 to H.264 Transcoding," *Proceedings of the ICCE*, pp. 419–420, Jan. 2006.

[63] G. Chen, S. Lin, and Y. Zhang, "A fast Coefficients Conversion Method for the Transform Domain MPEG-2 to H.264 Transcoding," *International Conference on Digital Telecommunications*, vol. 29, no. 30, pp. 17–20, Aug. 2006.

[64] X. Lu, A.M. Tourapis, P. Yin and J. Boyce, "Fast Mode Decision and Motion Estimation for H.264 With A Focus on MPEG-2 to H.264 Transcoding," *IEEE International Symposium on Circuits and Systems*, pp. 1246–1249, May 2005.

[65] G. Fernández, P. Cuenca, L. O. Barbosa, and H. Kalva, "Very low complexity mpeg-2 to h.264 transcoding using machine learning," pp. 931–940, Oct. 2006.

[66] B. Hu, P. Zhang, Q. Huang and W. Gao, "Reducing Spatial Resolution for MPEG-2 to H.264/AVC Transcoding," *IEEE Pacific-rim Conference on Multimedia*, pp. 830–840, Oct. 2005.

[67] T.D.Nguyen, G.S.Lee, J.Y.Chang and H.J.Cho, "Efficient MPEG-4 to H.264/AVC Transcoding with Spatial Downscaling," *ETRI Journal*, vol. 29, no. 6, pp. 826–828, Dec. 2007.

[68] Y.-P.Tan and H. Sun, "Fast Motion Re-estimation for Arbitrary Downsizing Video Transcoding Using H.264/AVC Standard," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 3, pp. 887–889, Aug. 2004.

[69] Haiyan Shu, "An Efficient Arbitrary Downsizing Algorithm for Video Transcoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 6, pp. 887–889, Jun. 2004.

[70] Z. Zhou, S. Sun, S. Lei and M.T. Sun, "Motion Information and Coding Mode Reuse for MPEG-2 to H.264 Transcoding," *IEEE International Symposium on Circuits and Systems*, pp. 1230–1233, May 2005.

[71] F. Pan, X. Lin, R. Susanto, K.P. Lim, Z.G. Li, G.N. Feng, D.J. Wu and S. Wu, "JVT-G013: Fast Mode Decision for Intra Prediction," *ISO/IEC MPEG and ITU-T VCEG Joint Video Team*, Mar. 2003.

[72] K.P. Lim, S. Wu, D.J. Wu, S. Rahardja, X. Lin, F. Pan and Z.G. Li, "JVT-I020: Fast Inter Mode Selection," *ISO/IEC MPEG and ITU-T VCEG Joint Video Team, (San Diego)*, Sep. 2003.

[73] Y. Su, J. Xin, A. Vetro and H. Sun, "Efficient MPEG-2 to H.264/AVC Intra Transcoding in Transform-domain," *IEEE International Symposium on Circuits and Systems*, pp. 1234–1237, May 2005.

[74] J. Xin, A. Vetro and H. Sun, "Efficient Macroblock Coding Mode Decision for H.264/AVC Video Coding," *in Proceedings of the 24th Picture Coding Symposium (PCS '04), San Francisco, Calif, USA,*, pp. 53–58, Dec. 2004.

[75] "MPEG-2 Encoder/Decoder V1.2," 1996. [Online]. Available: http://www.mpeg.org/MPEG/MSSG

[76] C. Lucarz, M. Mattavelli, J. Thomas-Kerr, and J. Janneck, "Reconfigurable Media Coding: A New Specification Model for Multimedia Coders," *IEEE Workshop on Signal Processing Systems*, pp. 481–486, Oct. 2007.

[77] ISO/IEC 23001-5, "Bitstream Syntax Description Language," Oct. 2007.

[78] ISO/IEC14496, "Coding of Audio-visual Objects (MPEG-4)," 2004.

[79] J. Eker and J. Janneck, "CAL Language Report," *Technical Memo UCB/ERL M03/48, University of California at Berkeley*, Dec. 2003.

[80] [Online]. Available: http://ptolemy.eecs.berkeley.edu

[81] "The Open DataFlow Environment on Sourceforge." [Online]. Available: http://opendf.sourceforge.net/

[82] W. De Neve, F. De Keukelaere, K. De Wolf and R. Van de Walle, "Applying MPEG-21 BSDL to the JVT H.264/AVC specification in MPEG-21 Session Mobility scenarios," *5th*

International Workshop on Image Analysis for Multimedia, Published on CD-ROM, Apr. 2004.

[83] G. Roquier, M. Wipliez, M. Raulet, J. Janneck, I. Miller, and D. Parlour, "Automatic software synthesis of dataflow program: An MPEG-4 simple profile decoder case study," *IEEE Workshop on Signal Processing Systems*, pp. 281–286, Jul. 2008.

[84] J. Janneck, I. Miller, D.B. Parlour, M. Mattavelli, C. Lucarz, M. Wipliez, M. Raulet and G. Roquier, "Translating Dataflow Programs to Efficient Hardware: an MPEG-4 Simple Profile Decoder Case Study," *Design, Automation and Test in Europe (DATE08)*, 2008.

[85] E. S. Jang, J. Ohm, and M. Mattavelli, "Whitepaper on Reconfigurable Video Coding (RVC)," *ISO/IEC JTC1/SC29/WG11*, Jan. 2008.

[86] D. Ding, H. Qi, L. Yu, T. Huang, and W. Gao, "Reconfigurable Video Coding Framework and Decoder Reconfiguration Instantiation of AVS," *Signal Processing: Image Communication*, vol. 24, no. 4, pp. 287–299, Jul. 2009.

[87] H. Aman-Allah, K. Maarouf, E. Hanna, I. Amer, and M. Mattavelli, "CAL Dataflow Components for an MPEG RVC AVC Baseline Encoder," *Journal of Signal Processing Systems for Signal, Image and Video Technology*, published online on 25th Jul. 2009. DOI:10.1007/s11265-009-0396-6.

[88] *SMPTE 314M-1999 Data Structure for 25 and 50 MB/s*, SMPTE, 1999.

[89] *SMPTE 370M-2006 Data Structure for 100 MB/s*, SMPTE, 2006.

[90] [Online]. Available: http://www.tik.ee.ethz.ch/~moses/

[91] [Online]. Available: http://ffmpeg.org/

[92] J. Janneck, I. Miller, D. Parlour, G. Roquier, M. Wipliez, and M. Raulet, "Synthesizing Hardware from Dataflow Programs," *Journal of Signal Processing Systems for Signal, Image and Video Technology*, pp. 287–292, published online on 11th Jul. 2008. DOI:10.1007/s11265-009-0397-5.

Appendix A

List of Publications & Contributions

↕ **Journals:**

1. Jianjun Li and Esam Abdel-Raheem, "Fast Implementation of H.264 4x4 Intra Prediction", IEICE Electronics Express, vol. 7, no. 5, pp. 332-338, Mar. 2010.
2. Jianjun Li and Esam Abdel-Raheem, "Modeling DV/DVCPRO Standards on Reconfigurable Video Coding Framework", Accepted by Journal of Electrical and Computer Engineering, Hindawi Publishing Corporation, Apr. 2010.
3. Jianjun Li and Esam Abdel-Raheem, "Efficient Rate Control for H.264/AVC Intra Frame", Submitted to IEEE Transactions on Consumer Electronics, Apr. 2010.
4. Jun Xin, Jianjun Li, Anthony Vetro, Huifang Sun and Shun-ichi Sekiguchi, "Motion mapping and mode decision for MPEG-2 to H.264/AVC transcoding", Multimedia Tools Appl.35(2): 203-223 (2007).

↕ **Conferences:**

1. Jianjun Li, Dandan Ding, Christophe Lucarz, Samuel Keller and Marco Mattavelli, "Efficient data flow variable length decoding implementation for the MPEG reconfigurable video coding framework", IEEE Workshop on Signal Processing Systems (SIPS), Oct. 2008.
2. Jianjun Li, Ahmadi, "Realizing high throughput transforms of H.264/AVC", The IEEE International Symposium on Circuits and Systems (ISCAS), May 2008.

3. Jun Xin, Jianjun Li, Anthony Vetro, Huifang Sun and Shun-ichi Sekiguchi, "Motion Mapping for MPEG-2 to H.264/AVC Transcoding", The IEEE International Symposium on Circuits and Systems (ISCAS), May 2007.

↕ **Contributions to MPEG RVC Standard:**

1. ISO/IEC JTC1/SC29/WG11 MPEG2008/M15163, Jan.2008, Antalya, Turkey
   "Auto-generation of RVC Parser from BSDL Syntax Description: Variable Length Decoding"
   Lucarz, Christophe, Li, Jianjun, Mattavelli, Marco and Ding, Dandan
2. ISO/IEC JTC1/SC29/WG11 MPEG2008/M15164, Jan.2008, Antalya, Turkey
   "Functional Units for RVC Toolbox: Variable Length Decoding"
   Lucarz, Christophe, Li, Jianjun, Mattavelli, Marco and Ding, Dandan
3. ISO/IEC JTC1/SC29/WG11 MPEG2008/M15159, Jan. 2008, Antalya, Turkey
   "BSDL Description of MPEG-4 SP and AVC BP Bit Stream Syntax for RVC Framework"
   Lucarz, Christophe, Ding, Dandan, Li, Jianjun and Mattavelli, Marco

Appendix B

Fast Intra4x4 Prediction

Table B.1: Definition of intra4x4 prediction modes.

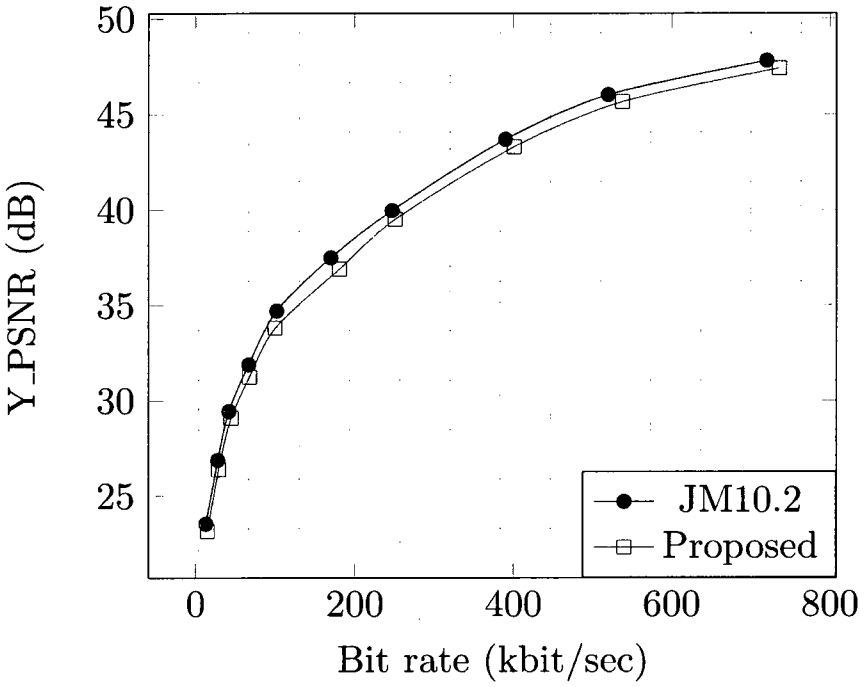
Mode	L	K	J	I	M	A	B	C	D	E	F	G	H	Equation
Vertical						1								A
							1							B
								1						C
									1					D
Horizontal	1													L
		1												K
			1											J
				1										I
DC	1	1	1	1		1	1	1	1					(I+J+K+L+ A+B+C+D+4)>>3
DDL[1]						1	2	1						(A+2B+C+2)>>2
							1	2	1					(B+2C+D+2)>>2
								1	2	1				(C+2D+E+2)>>2
									1	2	1			(D+2E+F+2)>>2
										1	2	1		(E+2F+G+2)>>2
											1	2	1	(F+2G+H+2)>>2
												1	3	(G+3H+2)>>2
DDR[2]	1	2	1											(L+2K+J+2)>>2
		1	2	1										(K+2J+I+2)>>2
			1	2	1									(J+2I+M+2)>>2
				1	2	1								(I+2M+A+2)>>2

					1	2	1							(M+2A+B+2)>>2
						1	2	1						(A+2B+C+2)>>2
							1	2	1					(B+2C+D+2)>>2
VR^3					1	1								(M+A+1)>>1
						1	1							(A+B+1)>>1
							1	1						(B+C+1)>>1
								1	1					(C+D+1)>>1
		1	2	1										(K+2J+I+2)>>2
			1	2	1									(J+2I+M+2)>>2
				1	2	1								(I+2M+A+2)>>2
					1	2	1							(M+2A+B+2)>>2
						1	2	1						(A+2B+C+2)>>2
							1	2	1					(B+2C+D+2)>>2
HD^4	1	1												(L+K+1)>>1
		1	1			1	1							(K+J+1)>>1
			1	1										(J+I+1)>>1
				1	1									(I+M+1)>>1
	1	2	1											(L+2K+J+2)>>2
		1	2	1										(K+2J+I+2)>>2
			1	2	1									(J+2I+M+2)>>2
				1	2	1								(I+2M+A+2)>>2
					1	2	1							(M+2A+B+2)>>2
						1	2	1						(A+2B+C+2)>>2
VL^5						1	1							(A+B+1)>>1
							1	1						(B+C+1)>>1
								1	1					(C+D+1)>>1
									1	1				(D+E+1)>>1
										1	1			(E+F+1)>>1
						1	2	1						(A+2B+C+2)>>2
							1	2	1					(B+2C+D+2)>>2
								1	2	1				(C+2D+E+2)>>2
									1	2	1			(E+2F+G+2)>>2
										1	2	1		(F+2G+H+2)>>2
HU^6	1													L
	1	1												(L+K+1)>>1
		1	1											(K+J+1)>>1

			1	1										(J+I+1)>>1
	3	1												(3L+K+2)>>2
	1	2	1											(L+2K+J+2)>>2
		1	2	1										(K+2J+I+2)>>2

[1]Diagonal down-left
[2]Diagonal down-right
[3]Vertical right
[4]Horizontal down
[5]Vertical left
[6]Horizontal up

(a). Coastguard QCIF Performance Comparison



(b). Foreman QCIF Performance Comparison

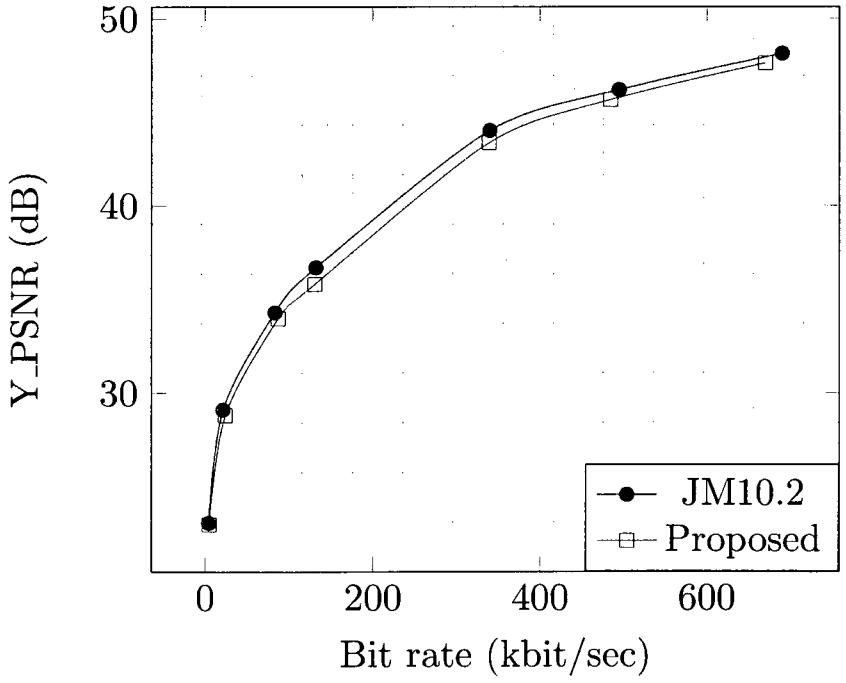
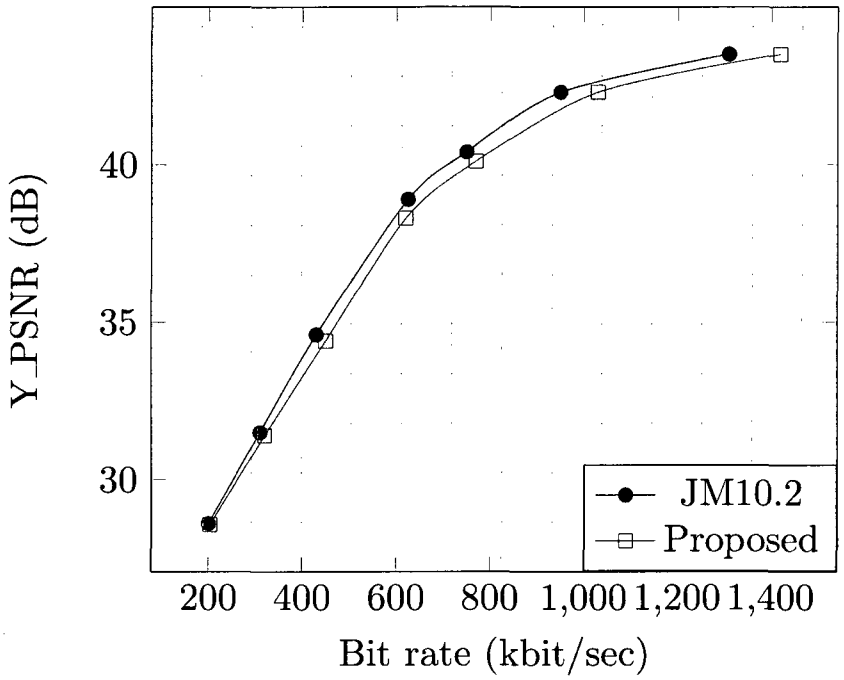


Figure B.1: Coastguard & Foreman Performance Comparison..

(a). Football CIF performance comparison



(b). Table tennis CIF performance comparison

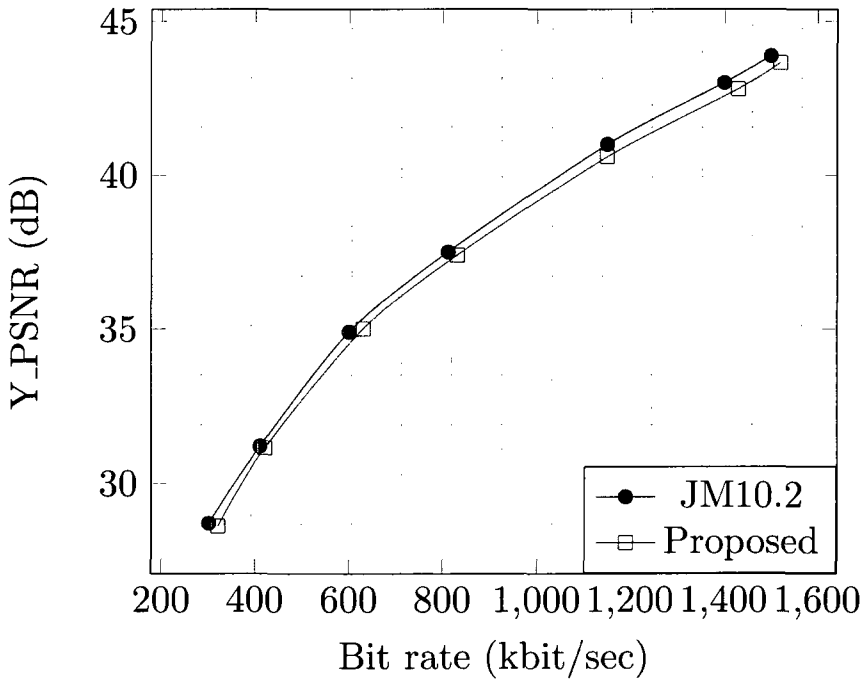


Figure B.2: Football & Table Tennis Performance Comparison.

Appendix C

Part of RVC-CAL Source Codes

C.1 Parser header RVC-CAL Source Code

```
=============================================================
actor ParseHeaders (
    // Maximum image width (in units of macroblocks) that the decoder can handle.
    // It is used to allocate line buffer space at compile time.
    int MAXW_IN_MB,

  int MB_COORD_SZ,
  int BTYPE_SZ,
  int MV_SZ,
  int NEWVOP,
  int INTRA,
  int INTER,
  int QUANT_MASK,
  int ROUND_TYPE,
  int FCODE_MASK,
  int FCODE_SHIFT,
  int ACPRED,
  int ACCODED,
  int FOURMV,
  int MOTION,
  int SAMPLE_COUNT_SZ,
  int SAMPLE_SZ
 ) bool bits, int(size=12) vld_data ==> string entab,int(size=20) vld_add, int(size=BTYPE_SZ) BTYPE, int(size=MV_SZ) M

  // Constants for various field lengths (bits) and special values.

  int VO_HEADER_LENGTH      = 27;
  int VO_NO_SHORT_HEADER    = 8;
  int VO_ID_LENGTH          = 5;

  int VOL_START_CODE_LENGTH              = 28;
  int VOL_START_CODE                     = 18;
  int VOL_ID_LENGTH                      = 5;
  int VIDEO_OBJECT_TYPE_INDICATION_LENGTH = 8;
  int VISUAL_OBJECT_LAYER_VERID_LENGTH    = 4;
  int VISUAL_OBJECT_LAYER_PRIORITY_LENGTH = 3;
  int ASPECT_RATIO_INFO_LENGTH            = 4;
  int ASPECT_RATIO_INFO_IS_DETAILED       = 15;
  int PAR_WIDTH_LENGTH                    = 8;
  int PAR_HEIGHT_LENGTH                   = 8;
  int CHROMA_FORMAT_LENGTH                = 2;
```

```
  int LOW_DELAY_LENGTH                    = 1;
  int FIRST_HALF_BIT_RATE_LENGTH          = 15;
  int LAST_HALF_BIT_RATE_LENGTH           = 15;
  int FIRST_HALF_VBV_BUF_SZ_LENGTH        = 15;
  int LAST_HALF_VBV_BUF_SZ_LENGTH         = 3;
  int FIRST_HALF_VBV_OCC_LENGTH           = 11;
  int LAST_HALF_VBV_OCC_LENGTH            = 15;
  int VOL_SHAPE_LENGTH                    = 2;
  int MARKER_LENGTH                       = 1;
  int TIME_INC_RES_LENGTH                 = 16;
  int VOL_WIDTH_LENGTH                    = 13;
  int VOL_HEIGHT_LENGTH                   = 13;
  int RUN_LENGTH                          = 6;
  int RUN_MASK                            = lshift( 1, RUN_LENGTH   ) - 1;
  int LEVEL_LENGTH                        = 12;
  int LEVEL_MASK                          = lshift( 1, LEVEL_LENGTH ) - 1;
  int MISC_BIT_LENGTH = 9;
  int VOP_START_CODE_LENGTH           = 32;
  int VOP_START_CODE                  = 438;
  int VOP_PREDICTION_LENGTH           = 2;
  int B_VOP                           = 2;
  int P_VOP                           = 1;
  int I_VOP                           = 0;
  int INTRA_DC_VLC_THR_LENGTH         = 3;
  int VOP_FCODE_FOR_LENGTH            = 3;
  int VOP_FCODE_FOR_MASK              = lshift( 1, VOP_FCODE_FOR_LENGTH ) - 1;
  int BITS_QUANT                      = 5;
  int BITS_QUANT_MASK                 = lshift( 1, BITS_QUANT           ) - 1;
  int MCBPC_LENGTH                    = 9;
  int ESCAPE_CODE                     = 7167;

  function mask_bits( v, n ) --> int :
    bitand( v, lshift(1,n)-1 )
  end

  // Utility action to read a specified number of bits.
  // This is an unnamed action, ie it is always enabled and has highest priority.
  // Use the procedures set_bits_to_read() to start reading, test for
  // completion with the boolean done_reading_bits() and get the value
  // with read_result(). Use the done function in a guard to wait for the
  // reading to be complete.
  int(size=7) bits_to_read_count := -1;
  int(size=33) read_result_in_progress;
  procedure set_bits_to_read( int count )
  begin
    bits_to_read_count := count - 1;
    read_result_in_progress := 0;
  end
  function done_reading_bits() --> bool : bits_to_read_count < 0 end
  function read_result() --> int : read_result_in_progress end
  action bits:[ b ] ==>
  guard
    not done_reading_bits()
  do
    read_result_in_progress := bitor( lshift( read_result_in_progress, 1), if b then 1 else 0 end );
    bits_to_read_count := bits_to_read_count - 1;
    bit_count := bit_count + 1;
  end

  int(size=4) bit_count := 0;

/************************************************************
 ************************************************************
 ********                start VOL                  ********
 ************************************************************
 ************************************************************/

  look_for_vo: action ==>
```

```
do
  set_bits_to_read( VO_HEADER_LENGTH );
end

// We can only handle VOL without short header
vo_header.good: action ==>
guard
  done_reading_bits(),
  mask_bits( read_result(), VO_HEADER_LENGTH ) = VO_NO_SHORT_HEADER
do
  set_bits_to_read( VO_ID_LENGTH );
end

vo_skip_id_field: action ==>
guard
  done_reading_bits()
do
  set_bits_to_read( VOL_START_CODE_LENGTH );
end

vol_startcode.good: action ==>
guard
  done_reading_bits(),
  mask_bits( read_result(), VOL_START_CODE_LENGTH ) = VOL_START_CODE
do
  // Ignore the next two fields
  set_bits_to_read( VOL_ID_LENGTH + VIDEO_OBJECT_TYPE_INDICATION_LENGTH );
end

vol_object_layer_identification: action bits:[b] ==>
guard
  done_reading_bits()
do
  set_bits_to_read(
     if b then
       // is_object_layer_identifier asserted
       VISUAL_OBJECT_LAYER_VERID_LENGTH + VISUAL_OBJECT_LAYER_PRIORITY_LENGTH + ASPECT_RATIO_INFO_LENGTH
     else
       ASPECT_RATIO_INFO_LENGTH
     end );
  bit_count := bit_count + 1;
end

vol_aspect.detailed: action  ==>
guard
  done_reading_bits(),
  mask_bits( read_result(), ASPECT_RATIO_INFO_LENGTH ) = ASPECT_RATIO_INFO_IS_DETAILED
do
  // Skip over aspect ratio details
  set_bits_to_read( PAR_WIDTH_LENGTH + PAR_HEIGHT_LENGTH );
end

vol_control.detailed: action bits:[b] ==>
guard
  done_reading_bits(),
  b
do
  set_bits_to_read( CHROMA_FORMAT_LENGTH + LOW_DELAY_LENGTH );
  bit_count := bit_count + 1;
end

vol_vbv.detailed: action bits:[b] ==>
guard
  done_reading_bits(),
  b
do
  set_bits_to_read(   FIRST_HALF_BIT_RATE_LENGTH    + MARKER_LENGTH +
                      LAST_HALF_BIT_RATE_LENGTH     + MARKER_LENGTH +
```

```
                    FIRST_HALF_VBV_BUF_SZ_LENGTH  + MARKER_LENGTH +
                    LAST_HALF_VBV_BUF_SZ_LENGTH   +
                    FIRST_HALF_VBV_OCC_LENGTH     + MARKER_LENGTH +
                    LAST_HALF_VBV_OCC_LENGTH      + MARKER_LENGTH );
  bit_count := bit_count + 1;
end

vol_shape: action ==>
guard
  done_reading_bits()
do
  set_bits_to_read( VOL_SHAPE_LENGTH + MARKER_LENGTH + TIME_INC_RES_LENGTH + MARKER_LENGTH + 1 );
end

int(size=7) mylog;

vol_time_inc_res: action ==>
guard
  done_reading_bits()
var
  int(size=TIME_INC_RES_LENGTH+1) time_inc_res := mask_bits( rshift( read_result(), 2 ), TIME_INC_RES_LENGTH ),  //
  int(size=7) count := 0,
  int(size=7) ones  := 0
do
  while ( count = 0 or time_inc_res != 0 ) do
    if bitand( time_inc_res, 1 ) = 1 then
      ones := ones + 1;
    end
    count := count + 1;
    time_inc_res := rshift( time_inc_res, 1 );
  end
  mylog := if ones > 1 then count else count - 1 end;
  mylog := if mylog < 1 then 1 else mylog end;
  set_bits_to_read(if bitand( read_result(), 1 ) = 1 then
    // fixed vop rate
    //mylog + MARKER_LENGTH
    mylog
  else
    // variable vop rate
    //MARKER_LENGTH
    0
  end + MARKER_LENGTH + VOL_WIDTH_LENGTH + MARKER_LENGTH );
end

// The vol width and height in units of macroblocks, ie. the pixel w/h divided by 16.
// Note: there is no provision in the model for pixel sizes that are no multiples of 16.
int(size=MB_COORD_SZ) vol_width;
int(size=MB_COORD_SZ) vol_height;

vol_width: action ==>
guard
  done_reading_bits()
do
  vol_width  := mask_bits( rshift( read_result(), MARKER_LENGTH + 4 ), VOL_WIDTH_LENGTH-4 ); // strip marker and div
  set_bits_to_read( VOL_HEIGHT_LENGTH + MARKER_LENGTH );
end
```

```
schedule fsm look_for_vo :

    // Process a new VOL
    look_for_vo        ( look_for_vo                    ) --> vo_header;
    vo_header          ( generic_done                   ) --> stuck;
    vo_header          ( vo_header.good                 ) --> vo_skip_id_field;
    vo_skip_id_field   ( vo_skip_id_field               ) --> look_for_vol;

    // Process a new VOL
    look_for_vol       ( generic_done                   ) --> stuck;
    look_for_vol       ( vol_startcode.good             ) --> vol_object;
    vol_object         ( vol_object_layer_identification ) --> vol_aspect;
    vol_aspect         ( vol_aspect.detailed            ) --> vol_control;
    vol_aspect         ( generic_done                   ) --> vol_control;
    vol_control        ( vol_control.detailed           ) --> vol_vbv;
    vol_control        ( generic_done_with_bitread      ) --> vol_shape;
    vol_vbv            ( vol_vbv.detailed               ) --> vol_shape;
    vol_vbv            ( generic_done_with_bitread      ) --> vol_shape;
    vol_shape          ( vol_shape                      ) --> vol_time_inc_res;
    vol_time_inc_res   ( vol_time_inc_res               ) --> vol_width;
    vol_width          ( vol_width                      ) --> vol_height;
    vol_height         ( vol_height                     ) --> vol_misc;
    // vol_misc           ( vol_misc.unsupported           ) --> stuck;
    vol_misc           ( generic_done                   ) --> look_for_vop;

    // Process a new VOP
    look_for_vop       ( byte_align                     ) --> get_vop_code;
    get_vop_code       ( get_vop_code                   ) --> vop_code;
    vop_code           ( vop_code.done                  ) --> look_for_vol;
    vop_code           ( vop_code.start                 ) --> vop_predict;
    vop_code           ( generic_done                   ) --> stuck;
    vop_predict        ( vop_predict /* .supported */   ) --> vop_timebase;
    // vop_predict        ( generic_done                   ) --> stuck;
    vop_timebase       ( vop_timebase.one               ) --> vop_timebase;
    vop_timebase       ( vop_timebase.zero              ) --> vop_coding;
    vop_coding         ( vop_coding.uncoded             ) --> look_for_vop;
vop_coding         ( vop_coding.coded               ) --> send_new_vop_info;
send_new_vop_info  ( send_new_vop_cmd               ) --> send_new_vop_width;
send_new_vop_width ( send_new_vop_width             ) --> send_new_vop_height;
send_new_vop_height( send_new_vop_height            ) --> mb;

    // Start MB
    mb          ( mb_done                  ) --> look_for_vop;
mb            ( mcbpc_pvop_uncoded       ) --> pvop_uncoded1;
pvop_uncoded1 ( mcbpc_pvop_uncoded1      ) --> pvop_uncoded2;
pvop_uncoded2 ( mcbpc_pvop_uncoded1      ) --> pvop_uncoded3;
pvop_uncoded3 ( mcbpc_pvop_uncoded1      ) --> pvop_uncoded4;
pvop_uncoded4 ( mcbpc_pvop_uncoded1      ) --> pvop_uncoded5;
pvop_uncoded5 ( mcbpc_pvop_uncoded1      ) --> mb;
    mb          ( get_mcbpc                ) --> get_mbtype;
    get_mbtype  ( vld_failure              ) --> stuck;
    get_mbtype  ( get_mbtype               ) --> final_cbpy;
    final_cbpy  ( vld_failure              ) --> stuck;
    final_cbpy  ( final_cbpy_intra         ) --> block;
    final_cbpy  ( final_cbpy_inter         ) --> mv;

    // process all blocks in an MB
    block      ( mb_dispatch_done          ) --> mb;
    block      ( mb_dispatch_intra         ) --> texture;
    block      ( mb_dispatch_inter_ac_coded ) --> texture;
    block      ( mb_dispatch_inter_no_ac   ) --> block;

    // Start texture
    texture    ( vld_start_intra           ) --> get_dc_bits;
    texture    ( vld_start_inter           ) --> texac;
    get_dc_bits ( get_dc_bits.none         ) --> texac;
    get_dc_bits ( get_dc_bits.some         ) --> get_dc;
    get_dc_bits ( vld_failure              ) --> stuck;
```

```
    get_dc      ( dc_bits_shift            ) --> get_dc_a;
    get_dc_a    ( get_dc                   ) --> texac;
    texac       ( block_done               ) --> block;
    texac       ( dct_coeff                ) --> vld1;
    vld1        ( vld_code                 ) --> texac;
    vld1        ( vld_level                ) --> vld4;
    vld1        ( vld_run_or_direct        ) --> vld7;
    vld7        ( vld_run                  ) --> vld6;
    vld7        ( vld_direct_read          ) --> vld_direct;
    vld1        ( vld_failure              ) --> stuck;
    vld_direct  ( vld_direct               ) --> texac;
    vld4 ( do_level_lookup           ) --> vld4a;
    vld4 ( vld_failure                ) --> stuck;
    vld4a ( vld_level_lookup           ) --> texac;
    vld6 ( do_run_lookup              ) --> vld6a;
    vld6 ( vld_failure                ) --> stuck;
    vld6a ( vld_run_lookup             ) --> texac;

    // mv()
    mv              ( mvcode_done       ) -->  block;
    mv              ( mvcode            ) -->  mag_x;
    mag_x           ( vld_failure       ) -->  stuck;
    mag_x           ( mag_x             ) -->  get_residual_x;
    get_residual_x  ( get_residual_x    ) -->  mv_y;
    mv_y            ( mvcode            ) -->  mag_y;
    mag_y           ( vld_failure       ) -->  stuck;
    mag_y           ( mag_y             ) -->  get_residual_y;
    get_residual_y  ( get_residual_y    ) -->  mv;

    // stuck( stuck ) --> stuck_for_good;
  end

  priority

    vo_header.good         >  generic_done;
    vol_start_code.good    >  generic_done;
    vol_aspect.detailed    >  generic_done;
    vol_control.detailed   >  generic_done_with_bitread;
    vol_vbv.detailed       >  generic_done_with_bitread;
    // vol_misc.unsupported   >  generic_done;

    vop_code.done          >  generic_done;
    vop_code.start         >  generic_done;
    // vop_predict.supported  >  generic_done;
    vop_timebase.one       >  vop_timebase.zero;
    vop_coding.uncoded     >  vop_coding.coded;

    mb_done > get_mcbpc;
    mb_done > mcbpc_pvop_uncoded;
    get_mcbpc.pvop > mcbpc_pvop_uncoded;

    get_mbtype.noac > get_mbtype.ac;
    final_cbpy_inter > final_cbpy_intra;
    mb_dispatch_done > mb_dispatch_intra >
    mb_dispatch_inter_no_ac > mb_dispatch_inter_ac_coded;

    vld_start_intra > vld_start_inter;
    get_dc_bits.none > get_dc_bits.some;
    block_done > dct_coeff;
    vld_code > vld_level > vld_run_or_direct;
    vld_run > vld_direct_read;

    vld_start_inter.ac_coded > vld_start_inter.not_ac_coded;

    mvcode_done > mvcode;
  end

================================================================
```

C.2 CAL Source Code for VLD Function Unit

```
================================================================================
import all caltrop.lib.BitOps;
actor VLD_mcbpc_intra(int VLD_DATA_SZ, int VLD_ADDR_SZ)
  string bits ==> int(size=2) finish, int(size=VLD_DATA_SIZE) data:
  int START_INDEX = 0;
  int( size=VLD_ADDR_SZ ) vld_index;
  int( size=VLD_DATA_SZ ) vld_codeword := 1;
// ********** automatically generated part ********
list( type:int( size=VLD_DATA_SZ ), size=16 )
vld_table = [ 10, 12, 18, 58, 26, 76, 34, 16, 42, 50, 1, 80, 144, 208, 140, 204 ];
// ***********************************************
  procedure start_vld_engine( int index )
  begin
    vld_index := index;
    vld_codeword := 2;
  end
  function vld_success() --> bool: bitand(vld_codeword,3) = 0 end
  function vld_continue() --> bool: bitand(vld_codeword,3) = 2 end
  function vld_failure() --> bool: bitand(vld_codeword,1) = 1 end
  function vld_result() --> int( size=VLD_DATA_SZ ):
  rshift(vld_codeword,2) end

  start_VLD: action ==>
  do
    start_vld_engine( START_INDEX );
  end

  read_in_bits: action bits:[b] ==>
  do
   vld_codeword := vld_table[ vld_index + if b="1" then 1 else 0 end ];
   vld_index := rshift(vld_codeword,2);
  end
  continue_VLD: action ==> finish:[f]
  guard
     vld_continue()
  var
     int(size=2) f := 0
  end

  fail_VLD: action ==>
  guard
     vld_failure()
  do
     println("VLD FAILURE");
  end

  finish_VLD: action ==> finish:[f], data:[d]
  guard
     vld_success()
  var
     int(size=2) f := 2,
     int(size=VLD_DATA_SZ) d := vld_result()
  end

  schedule fsm start_VLD:
     start_VLD       ( start_VLD    ) --> read_in_bits;
     read_in_bits    ( read_in_bits ) --> process;
     process         ( continue_VLD ) --> read_in_bits;
     process         ( fail_VLD     ) --> start_VLD;
     process         ( finish_VLD   ) --> start_VLD;
  endschedule
endactor
================================================================================
```

C.3 Source Code of the Automatically Generated Parser

```
=================================================================
DCT_Coeff.read: action      ==>

guard
  readDone()
end


DCT_Coeff.output: action ==> B16: [current]

do
  current := read_result_in_progress ;
end


DCT_Coeff.finish: action B16_f: [finish] ==>


guard
 finish

do
 setRead(M4V_NEXT_ELEMENT_LENGTH);
end


DCT_Coeff.notFinished: action B16_f: [finish] ==>
guard

 not finish

do
 setRead(M4V_VLC_LENGTH);
end


[...]


// Finite State Machine

Previous_state      (previous_action)       --> DCT_Coeff_exists;

DCT_Coeff_exists    (DCT_Coeff.read)        --> DCT_Coeff_output;

DCT_Coeff_output    (DCT_Coeff.output)      --> DCT_Coeff_result;

DCT_Coeff_result    (DCT_Coeff.notFinished) --> DCT_Coeff_exists;

DCT_Coeff_result    (DCT_Coeff.finish)      --> Next_state;

=================================================================
```

Appendix D

Vita

Li, jianjun received his bachelor degree from Information Engineering in Xi'an University of Electrical Science and Technology, China in 1990. He received his Master of Engineering (MaSc) degree in Electrical and Computer Engineering from University of Western Ontario (UWO), London, Canada in 2005. His research interests are video/image signal processing, standards, algorithm and implementation.