For further volumes:
http://www.springer.com/series/4748

Xiaohua Tian · Thinh M. Le · Yong Lian

Entropy Coders of the H.264/AVC Standard

Algorithms and VLSI Architectures

Springer

Dr. Xiaohua Tian
National University of Singapore
Dept. Electrical & Computer Engineering
4 Engineering Drive 3
117576 Singapore
Singapore
tianxiaohua@nus.edu.sg

Dr. Thinh M. Le
11115 Boulevard Cavendish
#907 St-Laurent
H4R 2M9 Canada
Canada
thinh.le@ieee.org

Prof. Dr. Yong Lian
National University of Singapore
Dept. Electrical & Computer Engineering
4 Engineering Drive 3
117576 Singapore
Singapore
eleliany@nus.edu.sg

ISSN 1860-4862
ISBN 978-3-642-14702-9 e-ISBN 978-3-642-14703-6
DOI 10.1007/978-3-642-14703-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2010933787

Cover design: WMXDesign GmbH, Heidelberg

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To my beloved Father, Mother, and Xiu.
Xiaohua Tian

To my wife, Du-Tram, and daughter, Nha-Minh for their love and support.
Thinh M. Le

To my wife, Dr. Bei Hu, and daughters, Vivian and Alicia, for their love and support."
Yong Lian

Preface

This work is an attempt to provide background of a typical video coding block and the associated design consideration and methodology for realizing such block in VLSI circuits. Readers may find this book helpful in how an algorithm can be mapped onto the corresponding circuits, and the trade-offs between implementing and not implementing an algorithm in hardware.

In this book, the entropy coding tools adopted by the H.264/AVC video compression standard are presented in both algorithms and VLSI architectures. The book has two parts. Part A includes a brief background on video compression, in which entropy compression is an influential and interdependent component; introduction to the CAVLC and CABAC algorithms and their respective realized VLSI architectures. Part B comprises a design flow for a typical CABAC coder, with performance complexity analysis, design methodology, design consideration, and power reduction techniques. A WISHBONE system bus interface is also integrated into the design to enhance the portability of the CABAC encoder in a SoC-based video coding system.

The material contained in this book was developed over several years of research, and most recently, a research project of the same name. Citations of references in the book are included at appropriate places. Readers are referred to leading journals and well known conference proceedings (*IEEE Transactions on Circuits and Systems for Video Technology*, *IEEE Transactions on Circuits and Systems, IEEE Transactions on Computers, IEEE Transactions on Multimedia; Proceedings of IEEE International Conference on Image Processing, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, Proceedings of International Conference on Application-specific Systems, Architectures and Processors*) as well as several other sources for advances in the field.

The authors are thankful to the various editors, especially the following individuals: Shyam Parikkal Krishnamurthy for various software simulations, Xi Jiang for assistance in chip layout, and Trang T.T. Do for the word processing tasks.

Singapore — Xiaohua Tian
St-Laurent, QC, Canada — Thinh M. Le
Singapore — Yong Lian

Abstract

The latest international video coding standard H.264/AVC has significantly improved coding performance and network friendliness. Two entropy coding tools: *Context-based Adaptive Variable Length Coder* (CAVLC) and *Context-based Adaptive Binary Arithmetic Coder* (CABAC) have been adopted in different profiles of the standard. In the Baseline and Extended profiles, CAVLC is adopted, while in the Main and higher profiles targeting high bit-rate and high definition services, CABAC is used. The *Rate-Distortion Optimization* (RDO) is another very important tool that significantly improves the coding performance of H.264/AVC standard.

The first part of this book covers the theory behind entropy coding and especially CABAC. The second part of this book comprises a design flow for a typical CABAC coder, with performance complexity analysis, design methodology, design consideration, and power reduction techniques. A WISHBONE system bus interface is also integrated into the design to enhance the portability of the CABAC encoder in a SoC-based video coding system.

Compared to all other reported designs, this design supports most complete CABAC functions, including RDO and context model selection. The design procedure is SoC-based, targeting low complexity, high coding efficiency, and ease of system integration. The design achieves complete encoding of residual, prediction, and control data with full support for RDO mode and negligible work left on the host for input data packaging; and low bandwidth on system bus compared to the reported designs.

A full-pipelined top-level architecture is adopted in the proposed CABAC encoder, and constant coding throughput of 1 bin/cycle is achieved in different coding configurations. An efficient context access scheme is proposed to reduce context RAM access frequency and remove context model access delay, including techniques of context line access and local buffering, and context memory reallocation. The context RAM access frequency of the proposed encoder is significantly lower than that of designs with single context model access scheme and the design with small context model cache allocated. Unlike most reported designs, RDO related operations are fully supported in this CABAC encoder, including (a) coding state backup and restoration of context model, coding interval, and coded syntax element (SE), and (b) coding rate counting and output of each RDO mode. Compared to the

only reference design that fully supports RDO, context memory size of this design is only 16.0% of the reference design, and the context state backup and restoration operation delay of P8×8 RDO coding mode is 15.5% and 16.6% of the reference design in P- and B-frame coding tests.

The encoder design achieves higher coding speed compared to most of the reported designs, and the encoding speed of 325 Mbin/second is suitable for the video applications of real-time CIF coding in full RDO-on mode and HDTV coding in RDO-off mode. It is a low power design with internal power reduction techniques applied including clock gating and significant reduction of memory accesses. The reported gate-level power consumption of the whole encoder, including SoC interface and memory, is only 0.79 mW at HDTV720p60 8.9 Mbps RDO-off mode coding.

It is necessary to support both CABAC and RDO in the high quality and high definition H.264/AVC applications; however, computation complexity is also significantly increased. Due to the serial coding nature of CABAC with strong data dependency and frequent memory access, it is not efficient to accelerate CABAC encoding by software optimization. Therefore, hardware acceleration of CABAC encoding is necessary in the high bit-rate real time video encoding.

Contents

List of Figures

List of Tables

List of Abbreviations

2-D LS	2-D Logarithmic Search
A	
ABS	Absolute Operation
ADD	Addition Operation
AMBA	Advanced Microcontroller Bus Architecture
AR	Arithmetic Coding and Renormalization
ASIC	Application Specific Integrated Circuit
AVC	Advanced Video Coding
B	
BAC	Binary Arithmetic Coding
BAD	Binary Arithmetic Decoding
BB	Bypass Bin
BM	Binarization Matching
BMA	Block Matching Algorithm
BN	Binarization
BP	Bit Packing (of Codewords)
BP_BPM	Band-Pass Bit Plane Matching
BPM	Bit Plane Matching
BPMB	4×4 Block, MB/Sub-MB Partition, or MB
C	
CA	Context Access
CABAC	Context-Based Adaptive Binary Arithmetic Coder
CAVLC	Context-Based Adaptive Variable Length Coder
CBF	Coded Block Flag
CBP	Coded Block Pattern
CCIR	Consultative Committee for International Radio
CDS	Conjugate Direction Search
CIF	Common Intermediate Format
CM	Context Modeling
CS	Context Model Selection

CtxIdx	Context Index
CtxIdxInc	Context Index Increment
CtxOffset	Context Offset

D

DCT	Discrete Cosine Transform
DPCM	Differential Pulse Code Modulation
DWT	Discrete Wavelet Transform

E

EBCOT	Embedded Block Coding with Optimized Truncation
EGk	kth-Order Exp-Golomb binarization
EOS	End of Slice

F

FBMA	Full-Search Block Matching Algorithm
FFT	Fast Fourier Transform
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
FWFT	First Word Fall Through

G

GOP	Group of Picture

H

HDTV	High Definition TV

I

IMBS	Inverse Multiple Branch Selection
INTERCON	WISHOBNE System Bus Inter-connection Block
IP	Intellectual Property
IT	Integer Transform

J

JBIG	Joint Bi-Level Image Experts Group
JPEG	Joint Photographic Expert Group

L

LPS	Least Probable Symbol
LSCF	Last Significant Coefficient Flag
LSZ	Least Significant Zero
LUT	Lookup Table
LZD	Leading Zero Detection

M

MAE	Mean Absolute Error
MB	Macroblock
MBAFF	MB-Level Adaptive Frame/Field Coding
MBIST	Memory Build In Self Test
MC	Motion Compensation
ME	Motion Estimation
ME/C	Motion Estimation/Compensation
MF	Motion-Field
MPEG	Moving Picture Expert Group
MPS	Most Probable Symbol
MSE	Mean Square Error
MV	Motion Vector
MVD	Motion Vector Difference

N

NAL	Network Abstraction Layer

O

OS	Outstanding

P

PAFF	Picture-Level Adaptive Frame/Field Coding
PD	Pixel Decimation
PDC	Pixel Difference Classification

Q

QCIF	Quarter Common Intermediate Format
QP	Quantization Parameter

R

RB	Regular Bin
RDO	Rate-Distortion Optimization
RGB	Red Green Blue
RLC	Run Length Coding

S

SA	Search Area
SCF	Significant Coefficient Flag
SE	Syntax Element
SES	Simple and Efficient Search
SIMD	Single Instruction Multiple Data
SLA	Symbol Look Ahead
SoC	System-on-Chip

T

TSS	Three-Step Hierarchical Search technique
TU	Truncated Unary Binarization

U

UVLC	Universal Variable Length Coding

V

VCEG	Video Coding Expert Group
VCL	Video Coding Layer
VLC	Variable Length Coding
VQ	Vector Quantization

W

WB	WISHBONE System Bus Standard and Specifications

Y

YUV	Luminance Component Y, "Reddish" Chrominance Component C_r, and "Bluish" Chrominance Component C_b

Part I
Background on Video Coding and Entropy Coding

Chapter 1
Introduction to Video Compression

1.1 Background on Video Compression

A video clip is composed of several moving images which have been recorded in time. In order to provide quality of service over the limited communication bandwidth, some compression is required. Compression is a redundancy reduction process where redundancies in visual data are identified and represented using much fewer binary digits or, hereafter bits. For instance, in a television program in which an anchor delivers news with her head and shoulders appeared on the static background, the only relevant and moving visual information are her head and shoulders, and the static background is unchanged for some time. Therefore, the moving head-and-shoulders and the static background can be coded with different compression strategies.

A camera phone taking a video having frame size of the *Quarter Common Intermediate Format* (QCIF) 176 × 144 points, 15 frames per second (fps) video rate, at a 4:2:0 sampling scheme using 24-bit RGB color, results in an uncompressed bit stream of 15 × 176 × 144 × 12 bps or nearly 4.57 Mbits per second (Mbps). Assuming the video is to be transmitted over the 3G communication network, this video has to be compressed to 2 Mbps in case of stationary (non-moving) users, and 384 Kbps in case of users moving at 3 Km/h. For the user to have good viewing experience, that is the display rate should be the same as the capture rate (after a brief delay of about 150 ms), the stationary user must have a video phone capable of compressing 4.57 to 2, or 2.3:1, and the moving user must have a video phone capable of compressing 4.57 to 0.384, or nearly 11.9:1.

The larger the video frame size and the higher the video frame rate, the more effort is required to compress visual data. Moreover, the smaller the communication bandwidth, the more effort is required to further compress visual data. By taking advantage of the following intrinsic characteristics in the video sequences, and applying the corresponding compression techniques, decent amount of information reduction can be achieved:

Spectral redundancy: This redundancy exists between different color planes or spectral bands. For example, a 24-bit color image is sampled using a 4:2:0 sampling scheme results in a compression of 2:1. Consequently, an average of 12 bits is used to represent a RGB picture element or *pixel*, hereafter.

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6_1,

Spatial redundancy: This redundancy exists within a 2-D image. The redundancy reduction techniques include the transform coding, such as *Discrete Cosine Transform* (DCT), *Discrete Wavelet Transform* (DWT), *Integer Transform* (IT); or in some proprietary coder, *Vector Quantization* (VQ).

Temporal redundancy: This redundancy exists in the temporal domain of many adjacent frames. Examples of inter-frame transforms include: 3-D DCT, Adaptive VQ, conditional replenishment, or the most popular technique, *Motion Estimation/Compensation* (ME/C).

Statistical redundancy: This redundancy remains in post-spatial-compression and post-temporal-compression. Such statistical redundancy reduction techniques are *Run Length Coding* (RLC), *Variable Length Coding* (VLC) which are further classified to Huffman and Arithmetic Coding, Context-based Adaptive VLC (CAVLC), and Context-based Adaptive Binary Arithmetic Coding (CABAC).

In the following sections, the redundancy reduction techniques will be presented. The term "redundancy reduction" is used interchangeably with the term "compression".

1.2 Spectral Redundancy Reduction Techniques

Spectral can be loosely associated with color bands. Color bands can be treated differently based on human eyes' responses to each color bands. For instance, green is better perceived than red, and subsequently, red is better perceived than blue.

The *Consultative Committee for International Radio* (CCIR) 601 defines the relationship among the *luminance* component, Y, and "reddish" and "bluish" *chrominance* components C_r and C_b, respectively; sampling using a 4:4:4 scheme, and digitally gamma-corrected 24-bit Red Green Blue (RGB) values, as follows:

$$Y = 0.299R + 0.587G + 0.114B \tag{1.1}$$

$$C_r = 0.5R - 0.419G - 0.081B \tag{1.2}$$

$$C_b = -0.169R - 0.331G + 0.5B \tag{1.3}$$

And, inversely:

$$R = Y + 1.402(C_r - 128) \tag{1.4}$$

$$G = Y - 0.34414(C_b - 128) - 0.71414(C_r - 128) \tag{1.5}$$

$$B = Y + 1.772(C_b - 128) \tag{1.6}$$

The YC_rC_b (or YUV) sampling schemes are summarized in Fig. 1.1 below:

In Fig. 1.1, an X represents the sampling grid of the Y component, while an O represents the sampling grid of the C_r or C_b component. In the 4:4:4 sampling

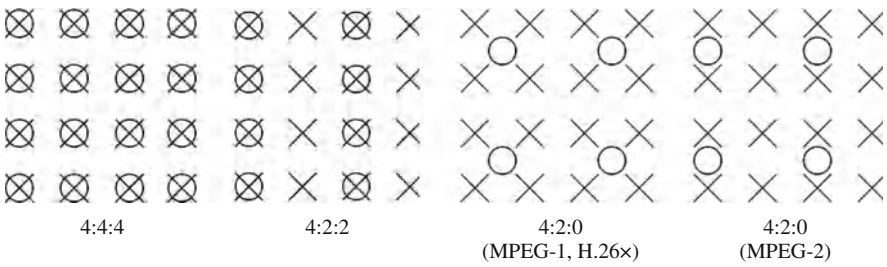

Fig. 1.1 YUV sampling schemes

scheme, for every X, there is one O for C_r, and one O for C_b. Note that the O of C_r and the O of C_b are co-located at the same point on the grid. Therefore, if 24 bits are used to represent 3 colors red, green, blue, the same 24 bits are used to represent Y, C_r, and C_b. No spectral compression is achieved.

In the 4:2:0 sampling scheme, for every 4Xs, there is one O for C_r, and one O for C_b. In other words, for every X, there is one quarter of O for C_r, and one quarter of O for C_b. If 24 bits are used to represent the 3 colors R, G, B, then only 12 bits are effectively assigned to Y, C_r, and C_b. Spectral compression is achieved by representing 24-bit grid blocks with 12 bits of color, or a 2:1 compression ratio.

1.3 Spatial Redundancy Reduction Techniques

Spatial domain compression techniques are referred to as *intra-frame* (I-frame) coding. The I-frame is coded independently without the knowledge from the preceding or succeeding frames. Predictive coding, scalar and vector quantization, transform coding, and entropy coding are common intra-frame coding techniques. Modern video coding standards employ most of them. Predictive coding and transform coding are discussed as follows.

The original predictive coding – widely used in voice communication systems – is known as *Differential Pulse Code Modulation* (DPCM). In intra-frame image coding, it explores the mutual redundancy among the neighbouring pixels. Rather than directly encoding the intensity of a pixel, its value is first predicted from the previously encoded pixels. Then the predicted pixel value is subtracted from the actual pixel value, and only the prediction error is encoded (instead of the absolute value).

From the frequency domain viewpoint, image data consist of low- and high-frequency coefficients. Human eyes are sensitive to low-frequency coefficients while high-frequency coefficients have less contribution to image quality. Hence, a transform coder transforms an image from the spatial domain to the frequency domain, and exploits the fact that a large amount of signal energy is concentrated in a small number of coefficients, at low frequencies if DCT is applied.

More precisely, an image can be partitioned into blocks, such as 4×4, 8×8, and 16×16, and transform coding is operated on the block-by-block basis. The

transformation matrix is a group of low-pass, band-pass, and high-pass filters which partition the power spectrum into distinct frequency bands. Most of the image signal energy, after transformation, stays within the low-frequency band, and concentrates in large coefficients at the upper left corner of the image block. On the other hand, high-frequency bands carry less signal energy, and their coefficients appear smaller in magnitude, and concentrate at the lower right corner of the block. After quantization, the smaller high-frequency band coefficients will effectively become zeros, and are discarded in entropy coding. Many transform algorithms have been proposed and reviewed in the following sections.

1.3.1 Discrete Cosine Transform (DCT)

DCT has been adopted as transform coder in the JPEG, H.26x, and MPEG-1 and -2 compression standards due to its excellent energy compaction for highly correlated data, and the availabilities of fast algorithms among the orthogonal transforms. Given a data sequence x(k), for k = 0, 1, 2, …, N–1. The 1-D N-point forward DCT X(n) is defined by:

$$X(n) = \frac{2}{N} C_n \sum_{k=0}^{N-1} x(k) \cos\left[\frac{\pi(2k+1)n}{2N}\right] \tag{1.7}$$

where n = 0, 1, …, N–1

And its inverse DCT, x(k), is computed as:

$$x(k) = \sum_{n=0}^{N-1} C_n X(n) \cos\left[\frac{\pi(2k+1)n}{2N}\right] \tag{1.8}$$

where k = 0, 1, …, N–1

$$C_n = \frac{1}{\sqrt{2}} \longrightarrow n = 0$$
$$C_n = 1 \longrightarrow n \neq 0$$

The first DCT algorithm was proposed by Ahmed et al. [Ahmed74] where a double-sized *Fast Fourier Transform* (FFT) algorithm was used with complex arithmetic throughout the computation. Chen et al. [Chen77] later presented a faster 1-D DCT computation (Fig. 1.2) by exploiting the sparseness of the matrices involved. For N=8, the numbers of additions and multiplications are 26 and 16, respectively. Lee [Lee84] further improved the technique by reducing the number multiplications to 12.

For image and video compression, 2-D DCT is required. Given a 2-D data sequence $x_{i,j}$, where i, j = 0, 1, 2, …, N–1. The N × N forward DCT $X_{k,l}$ is defined by:

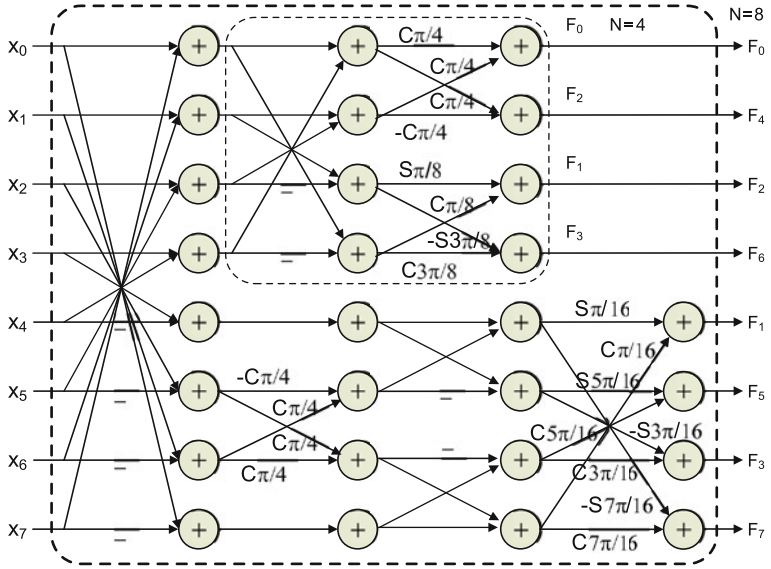

Fig. 1.2 Forward DCT flow graph for $N = 8$, Ci = cos i, Si = sin i [Chen77, Rao90]

$$X_{k,l} = \frac{2c_k c_l}{N}\left(\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} x_{i,j}\cos\frac{(2i+1)k\pi}{2N}\cos\frac{(2j+1)l\pi}{2N}\right) \tag{1.9}$$

The inverse DCT is defined by:

$$x_{k,l} = \left(\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} X_{k,l}\frac{2c_k c_l}{N}\cos\frac{(2i+1)k\pi}{2N}\cos\frac{(2j+1)l\pi}{2N}\right) \tag{1.10}$$

where $c_k = c_l = \dfrac{1}{\sqrt{2}}$, if $k, l = 0$ and $c_k = c_l = 1$, other wise.

By observing the pairs of Eqs. (1.7) & (1.8), and (1.9) & (1.10), it can be seen that the 2-D DCT operation is separable. Separability enables the transformation to be performed by taking a 1-D DCT on the rows of the input block, followed by a 1-D DCT on the columns of the semi-transformed matrix. This technique is sometimes referred to as row-column decomposition. The computational complexity is, therefore, 2 N times the numbers of additions and multiplications required for a 1-D DCT. For N=8, the number of additions and multiplications are 416 and 256, respectively, using Chen's algorithm [Chen77]; and 464 and 192, respectively, using Lee's [Lee84]. Later, Cho et al. [Cho91] proposed a 2-D DCT algorithm which required only 466 additions and 96 multiplications. In this algorithm, the 1-D DCT operations are first applied to the groups of odd-numbered samples and even-numbered

samples. The results of these operations then undergo some additions and shiftings (division by 2) before producing the final results.

In some applications, the DCT coefficients are scaled immediately after the transformation. The scaling factors (i.e. quantization coefficients) may be incorporated onto the matrices if no change to the quantization matrix is anticipated. Feig et al. [Feig92] proposed a 2-D scaled-DCT technique resulting in significant reduction in the number of additions and multiplications. For N=8, 462 additions and only 54 multiplications are needed. In theory, algorithms with the minimum number of multiplications are preferable. However, in practice, ease of implementation, regular structures and testability, as well as minimum buffer size are preferred.

1.3.2 Integer Transform (IT)

In the existing H.264/AVC video compression standard, the residual data will be coded using four different operations: 4 × 4 integer transform, 8 × 8 integer transform [Gordon04], 4 × 4 and 2 × 2 Hadamard transforms [Malva03]. The macroblocks with their scanning order are summarized and depicted in Fig. 1.3 [Richardson03]

The first type of transform is applied to 4 × 4 luma blocks 0–15 and 18–25. The second transform is applied to 8 × 8 blocks. The third and fourth types are applied to DC coefficients block –1, 16, and 17, when the macroblock is encoded in 16 × 16 intra-prediction mode.

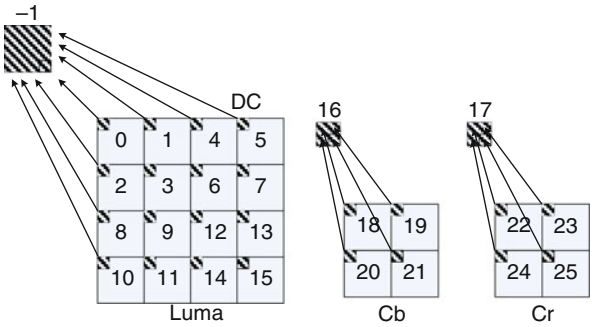

Fig. 1.3 Scanning order of residual blocks within a macroblock

1.3.2.1 The Integer Transform

$$Y = (C_4 X\, C_4^T) \odot E_f\,, \tag{1.11}$$

where C_4 and E_f are given by:

$$C_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \tag{1.12}$$

$$E_f = \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix} \tag{1.13}$$

and X is the input matrix.

The $\odot$ symbol indicates that each element of $(C_4\ X\ C_4^T)$ is multiplied by the scaling factor in the same position in the matrix E_f. C_4^T is the transpose of matrix C_4.

The 4×4 inverse integer transform can be written as:

$$X = C_4^i (Y \odot E_i)\, C_4^{iT} \tag{1.14}$$

where C_4^i and E_i are given by

$$C_4^i = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \tag{1.15}$$

$$E_i = \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \tag{1.16}$$

The 8×8 forward and inverse integer transforms can be performed in a similar manner [Richardson03].

The 4×4 Hadamard forward and inverse transforms are given by:

$$Y = (HXH^T)/2 \tag{1.17}$$

$$X = HYH^T \tag{1.18}$$

where:

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{1.19}$$

The 2 × 2 Hadamard transform use the same formula for forward and inverse:

$$Y = HXH^T \tag{1.20}$$

$$\text{with } H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{1.21}$$

1.3.2.2 Quantization and Rescaling

The scalar multiplication in the forward and inverse transforms can be absorbed into the scalar quantization step. The basic quantization operation for forward (1.22) and inverse (1.23) transform can be defined as follows:

$$Z_{ij} = round\,(Y_{ij}/Q_{step}) \tag{1.22}$$

$$Y_{ij} = Z_{ij}{}^{*}\ Q_{step} \tag{1.23}$$

There are 52 values of Q_{step} supported by H.264/AVC, and each has a *quantization parameter* (QP) associated with it.

To integrate the scalar multiplication by matrix E_f or $E_{i,}$ as shown above, and to avoid any division operations, the equation is changed to:

$$Z_{ij} = round\,(Y_{ij} \times (PF/\ Q_{step})) \tag{1.24}$$

$$Y_{ij} = Z_{ij} \times\ PF \times\ Q_{step} \times 64 \tag{1.25}$$

where:

$$PF/Q_{step} = MF/2^{qbits} \tag{1.26}$$

$$qbits = 15 + floor\,(QP/6) \tag{1.27}$$

And PF is the value from matrix E_f or E_i depending on the location of Y_{ij}.

Finally, (1.24) can be rewritten as follows:

$$\left|Z_{ij}\right| = ((\left|Y_{ij}\right| \times MF + f)) \gg qbits \tag{1.28}$$

$$sign\,(Z_{ij}) = sign\,(Y_{ij}) \tag{1.29}$$

The Offset parameter, f is $2^{qbits/3}$ for Intra Blocks or $2^{qbits/6}$ for Inter Blocks.

1.3.3 Discrete Wavelet Transform (DWT)

Wavelet Transform [Mallat89] offers a wide variety of useful features, in contrast to other transforms, such as Fourier Transform or Cosine Transform. These features include no blocking effect, lower aliasing distortion for signal processing applications, and inherent scalability. Due to the non-blocking property, Wavelet has been adopted for the JPEG2000 image compression standard.

DWT represents an arbitrary square integrable function[1] as superposition of a family of basis functions called *wavelets*. A family of wavelet basis functions can be generated by translating and dilating the mother wavelet corresponding to the family. The DWT coefficients can be obtained by taking the inner product between the input signal and the wavelet functions. Since the basis functions are translated and dilated versions of each other, a simpler algorithm, known as *Mallat's tree algorithm* or pyramid algorithm, was proposed [Mallat89]. In this algorithm, the DWT coefficients of one stage can be calculated from the DWT coefficients of the previous stage, which is expressed as follows:

$$W_H(n,j) = \sum_m W_L(m,j-1)g(m-2n) \tag{1.30}$$

$$W_L(n,j) = \sum_m W_L(m,j-1)h(m-2n) \tag{1.31}$$

where $W_L(p,q)$ is the pth scaling coefficient at the qth stage; $W_H(p,q)$ is the pth wavelet coefficient at the qth stage; and h(n) and g(n) are the dilation coefficients corresponding to the scaling and wavelet functions, respectively. Figure 1.4 shows this relationship.

When computing the DWT coefficients of the discrete-time data, it is assumed that the input data represent the DWT coefficients of a high-resolution stage. Equations (1.30) and (1.31) can then be used to obtain DWT coefficients of the subsequent stages. In practice, this decomposition is performed only for a few stages. The dilation coefficients h(n) also represent a low-pass filter, while those of g(n) represent a high-pass filter. Therefore, DWT extracts information from

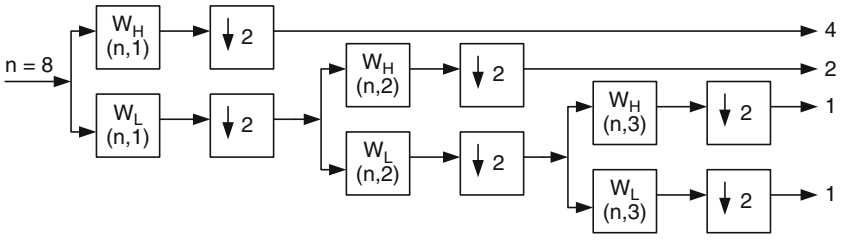

Fig. 1.4 3-stage 1-D DWT decomposition using pyramid algorithm [Mallat89]

[1] A function s(t) is square integrable if the following expression exists: $\int s^2(t)dt < \infty$

the signal at different scales. The first level of wavelet decomposition extracts the details of the signal (high-frequency components) while the second and subsequent wavelet decompositions extract progressively coarser information (lower-frequency components). In order to reconstruct the original data, the DWT coefficients are up-sampled and passed through another set of low- and high-pass filters, which is expressed as:

$$W_L(n,j) = \sum_k W_L(k,j+1)h'(n-2k) + \sum_l W_H(l,j+1)g'(n-2l) \qquad (1.32)$$

where h'(n) and g'(n) are respectively the low- and high-pass synthesis filter corresponding to the mother wavelet. It is observed from (1.32) that the jth level DWT coefficients can be obtained from (j+1)th level DWT coefficients.

Similar to the 2-D DCT, the 2-D DWT is also computed using row-column decomposition technique. In the first level of decomposition (Fig. 1.5), one low-pass sub-image (LL_2) and three orientation selective high-pass sub-images (LH_2, HL_2, and HH_2) are created. In the second level of decomposition, the low-pass sub-image LL_2 is further decomposed into one low- (LL_3) and three high-pass sub-images (LH_3, HL_3, and HH_3). This process is repeated on the low-pass sub-image to derive the higher level of decompositions. In other words, DWT decomposes an image into pyramidal structure of sub-images with various resolutions corresponding to the different scales. The inverse wavelet transform can be calculated in the reverse manner, i.e., starting from the lowest resolution sub-images, the higher resolution images are calculated recursively.

Unlike 2-D DCT where operations are performed on the non-overlapping $n \times n$ blocks where n is usually small, 2-D DWT requires operations on the entire image. Therefore, 2-D DWT can be classified as global operations. To efficiently implement

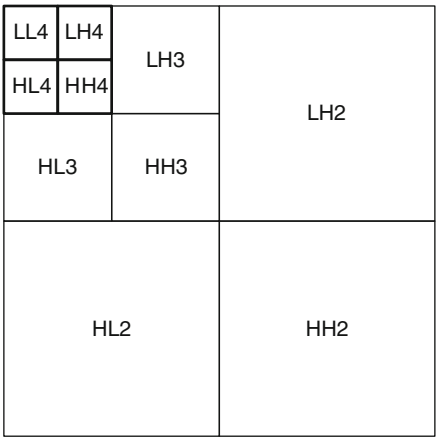

Fig. 1.5 Three-level 2-D wavelet decomposition

2-D DWT, either a 1-D architecture having a global data transposer, or a 2-D architecture is required.

1.4 Temporal Redundancy Reduction Techniques

Redundancy among the 2-D frames in the temporal dimension is said to be interframe transform coded. Such techniques can be Interframe Transform Coding such as 3-D DCT, Adaptive Vector Quantization; Conditional Replenishment; and Motion Estimation/Compensation (ME/C).

1.4.1 Interframe Transform Coding

By inspecting Eqs. (1.9) and (1.10), the separability property enables us to extend transform coding such as DCT to dimensions larger than two. As an example of a sequence of images, 3-D DCT can be implemented by a series of 1-D DCTs along each of the dimensions [Rao90]. Implementation of the 3-D DCT of a sequence of L frames on the $n \times n$ pixel blocks is performed as follows: First, the L-point 1-D transform is executed along the temporal direction, following by the 2-D transform of the frame data.

In video sequences, the statistics along the temporal dimension may vary significantly, therefore, adaptive techniques can substantially improve the coding performance [Habibi77]. For example, the effective number of bits assigned to each coefficient can be made proportional to the coefficient variance [Roese77]. Although transform coding can be extended to multiple dimensions, practical applications appear to be limited [Rao90].

1.4.2 Conditional Replenishment

The conditional replenishment technique [Netravali80] is based on dividing each frame into stationary and non-stationary parts. Only the changed parts are coded. If $x_{i,j,t}$ is a pixel at location (i, j) in the current frame (t), then the predicted value of $x_{i,j,t}$ is the reconstructed value $\hat{x}_{i,j,t-1}$ at the same spatial location in the previous frame $(t-1)$. The prediction error in this case is calculated using:

$$e_{i,j,t} = x_{i,j,t} - \hat{x}_{i,j,t-1} \tag{1.33}$$

If the magnitude of $e_{i,j,t}$ is greater than a pre-specified threshold, then it is quantized, coded, and transmitted along with the address (i, j) of $x_{i,j,t}$.

1.4.3 Motion Estimation

Motion estimation is a widely used technique to exploit the temporal correlation in a video sequence. Motion estimation attempts to obtain the motion information for the various regions/objects in a scene. Using motion estimation, high compression can be achieved by coding an $n \times n$ pixel block in the current frame by a motion vector (with respect to the best match block in a search area – SA- of the previous frame), and followed by the DCT coefficients of the estimated errors. There are two main classes of motion estimation/compensation algorithms: *pel-recursive* [Netravali89] and *block matching* (BMA's) [Jain81]. Pel-recursive algorithms evaluate the displacement of each pixel individually. These algorithms do not require the transmission of motion information but recursively use the luminance change to find the motion information. The advantage of a pel-recursive algorithm is the ability to overcome problems of multiple moving regions/objects as well as part of a region/object undergoing different displacements. The drawback of pel-recursive algorithms is the overwhelming information involved. Therefore, it is often operated in a predictive manner, that is the motion vectors between frame(t) and frame($t-1$) are predicted using the corresponding motion vectors between frame($t-1$) and frame($t-2$).

Block matching algorithms, on the other hand, assume that all pixels within a block have the same motion and behave well provided that the following conditions are met:

1. Zooming and rotation of objects are not considered. This is because the objects are assumed to move in translational mode in a plane that is parallel to the camera plane;
2. Pixel illumination between frames is spatially and temporally uniform;
3. Object displacement is constant within a small 2-D block of pixels; and
4. Matching distortion increases monotonically as the displaced candidate block moves away from the direction of the minimum distortion.

The most popular ME technique is the full-search block matching algorithm (FBMA) and is used as a benchmark to other simplified algorithms. FBMA searches all possible displaced locations within a $n \times n$ pixel SA to find the best match. Many objective matching criteria, MSE and MAE, to name a few, have been used to find the best match block [Chou89]. The MAE criterion is preferred because it requires no multiplication while achieving similar performance compared to the MSE.

1.4.3.1 Full-Search Block Matching Algorithm (FBMA)

The most popular ME technique is the *full-search block matching algorithm* (FBMA) [Jain81] and is used as a benchmark to other complexity reduction algorithms. FBMA searches exhaustively all possible displaced locations within an n $\times$ n pixel *search area* (SA) to find the best match. Many objective matching criteria, *Mean Square Error* (MSE) and *Mean Absolute Error* (MAE), to name a few, have

been employed to find the best match block [Chou89]. The MAE criterion is preferred because it requires no multiplication while achieving similar performance compared to the MSE. The FBMA is described as follows:

Let $X_{i,j}$ be the n × n pixel *reference block* located at coordinates (i, j), $Y_{i+k,\,j+l}$ be the n × n pixel *candidate block* at coordinates (i+k, j+l), and p be the maximum displacement (in steps of integer number of pixels). The search area is, therefore, of the size $(n+2p)^2$, while the number of search locations is $(2p+1)^2$. The MAE distortion measure is given by:

$$MAE_{k,l}(X,Y) = \frac{1}{n \times n} \sum_{i=1}^{n} \sum_{j=1}^{n} \left| X_{i,\,j} - X_{i+k,j+l} \right| \qquad -p \le k,l \le p \qquad (1.34)$$

The block matching process between any 2 consecutive frames is shown in Fig. 1.6.

In Fig. 1.6, the current frame(t) is divided into non-overlapping reference blocks. Each reference block in frame(t), is compared with the candidate blocks within a search area in the previous frame(t–1) in order to obtain the best match. Prior to searching, a reference coordinate is chosen. Usually, the center of the reference block and the center of the research area are chosen as reference coordinates. To simplify the search algorithm, the top-left corners can be chosen as reference coordinates as shown in Fig. 1.7.

In Fig. 1.7, the reference block A of size 16 × 16 is shown on the left, while the search area is shown on the right. The FBMA is described as follows:

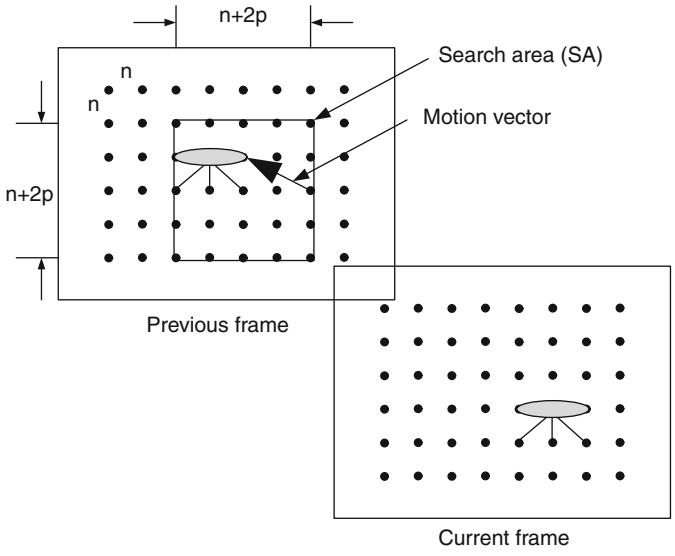

Fig. 1.6 The block matching process

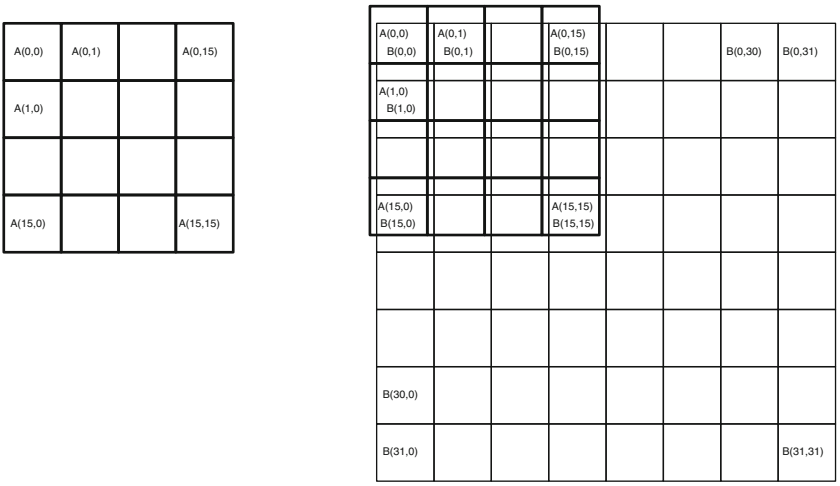

Fig. 1.7 Grid-based block matching process

- The reference block A is compared with the candidate block B, one-by-one by moving one pixel to the right across the columns, and one pixel down the rows, until a candidate block (which results in the least distortion) is found.
- Distortion at a particular displacement is calculated by summing all the 256 absolute differences of the respective pixels. In Eq. (1.34), the $1/(n^2)$ factor can be ignored since all distortions are divided by the same value n^2.

There are a total of $(16+1)^2 = 289$ search locations. The single location which results in the least distortion will be selected as the best match, and the (k,l) displacement between the reference block and the best match (candidate) block is used to represent the corresponding motion vector.

For each of the $(2p+1)^2$ search locations, the MAE computation requires $2n^2$ operations (one subtract-absolute operation, and one addition). Thus, for K reference macro-blocks, the computational complexity of FBMA is of order $O(2Kn^2(2p+1)^2)$. In a H.264/AVC QCIF video frame, namely size 176 × 144 pixels, $K=(176/16)\times(144/16)=99$, with $n=16$ and $p=8$, more than 27 million operations are required.

Motion estimation is well-known for its achievable visual data compression taking advantage of the trackable movement of moving objects/scences, yet it is also well-known for its highly intensive computation. Assuming that the factor $1/n^2$ is factored out, for each distortion computation shown in Fig. 1.8, the following operations are required:

$$\begin{aligned} \mathrm{MAE}_1 = &|\mathrm{A}(0,0) - \mathrm{B}(0,0)| + \ldots + |\mathrm{A}(0,15) - \mathrm{B}(0,15)| + \ldots + |\mathrm{A}(15,0) \\ &- \mathrm{B}(15,0)| + \ldots + |\mathrm{A}(15,15) - \mathrm{B}(15,15)| \end{aligned} \tag{1.35}$$

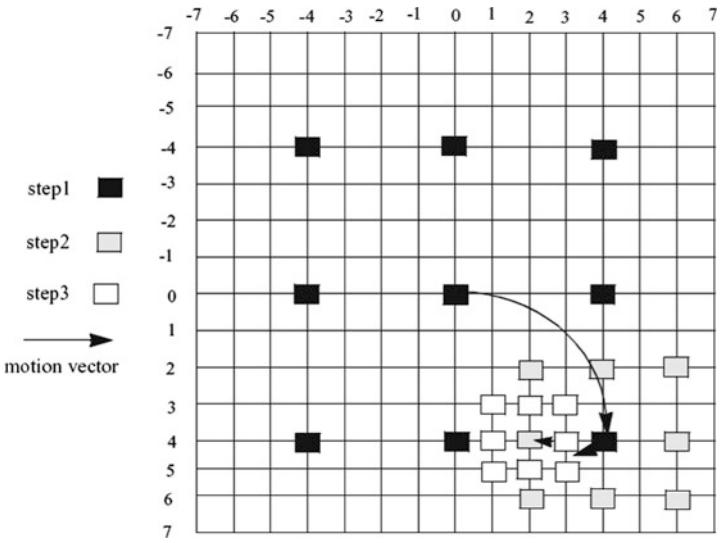

Fig. 1.8 Three-step search [Koga81]

To perform distortion computation, two general approaches are carried out:

- To compute MAE_1 or any MAE_i in a sequential manner, each "subtraction-absolute" (ABS) operation can be performed followed by a summation (ADD). Sequential computation is the least costly in terms of hardware resources, but it takes the longest time.
- To compute MAE_1 or any MAE_i in a parallel manner, 16 "subtraction-absolute" (ABS) operations can be performed in parallel followed by parallel summations (ADD). Note that any degree of parallelism can be architected. For example, 2, 4, 8, 16, 32, 64, 128, or 256 "subtraction-absolute" operations can be performed in parallel. The higher the degree of parallelism, the faster the operation, and consequently the more hardware is used, and the costlier the solution.

1.4.3.2 Group 1: Reduction in the Number of Search Locations

In the Three-Step hierarchical Search technique (TSS), Koga et al. [Koga81] have proposed a *divide-and-conquer* search method where the entire SA is first divided into 9 sub-SA's. After the first iteration, the sub-SA which results in the lowest MAE will be further divided and searched until a 3×3 search window is obtained, and the final motion vector is determined (Fig. 1.8).

The TSS algorithm is reported to achieve a speed-up of 9 while it suffers a loss of 0.5–1.5 dB in performance compared to FBMA [Lu97].

Lu et al. [Lu97] further optimized the TSS by making directional decision, referred to as Simple and Efficient Search (SES), based on half of TSS's search locations. There are two phases to SES. Phase one is to identify a search quadrant,

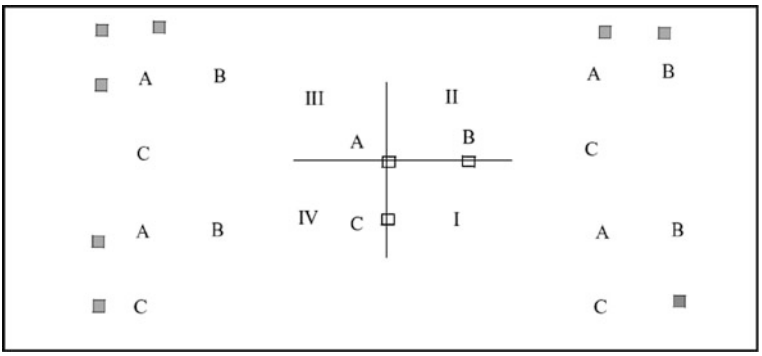

Fig. 1.9 Simple and efficient search

and phase two is to find the minimum error location in the selected quadrant. Let point A be the center of the SA (Fig. 1.9), points B and C (shown by clear boxes) be the search locations on the right of and below point A, respectively. The quadrants are numbered from I to IV in the counter clockwise direction. The procedure for identifying a search quadrant can be described as follows:

- If $MAE(A) \geq MAE(B)$ and $MAE(A) \geq MAE(C)$, quadrant I is selected;
- If $MAE(A) \geq MAE(B)$ and $MAE(A) < MAE(C)$, quadrant II is selected;
- If $MAE(A) < MAE(B)$ and $MAE(A) < MAE(C)$, quadrant III is selected; and
- If $MAE(A) < MAE(B)$ and $MAE(A) \geq MAE(C)$, quadrant IV is selected.

Once a quadrant is selected, on the average, two other locations are searched (solid boxes). Therefore, the number of search locations required in the first step is 5 (A, B, C, and 2 others); and the number of search locations needed in subsequent steps are 4 (new B, new C, and 2 others). Therefore, SES achieves a speed-up of nearly 2, while having a degradation of less than 0.1 dB compared to TSS.

In the 2-D Logarithmic Search (2-D LS), a moving window technique is applied. Five locations which contain the center of the search window and the four points corresponding to the North, South, East, and West directions will be searched. The search procedure starts from the center of the SA and proceeds to the N, S, E, and W directions, whichever results in the smallest distortion. Once the search focuses at a particular center, the step size between the search locations is reduced by a factor of 2. The search procedure is repeated until the search window is reduced to a 3×3 pixel window. In this final step, all 9 locations within the window are searched.

Finally, in Conjugate Direction Search (CDS), Srinivasan et al. [Srinivasan85] propose a multiple 1-D search. Starting at point A, the search proceeds, location by location, in the horizontal direction. Once the position associated with the lowest MAE is obtained, at point B, the search proceeds in the vertical direction. When the lowest MAE is obtained, at point C, the diagonal connecting A and C will then be searched. The best match location is assumed to lie on the line connecting AC.

1.4.3.3 Group 2: Reduction in the Number of Pixels Involved

Liu et al. [Liu93] have proposed two algorithms for block motion estimation that produce similar performance to that of FBMA with computation reduced by a factor of 8 or 16. The algorithms are based on *pixel decimation* (PD) and *motion-field* (MF).

PD is developed based on the assumption that all pixels in a block move by the same amount, therefore, a good estimate of the motion could be obtained by using only a fraction of the total number of pixels in that block. This technique employs every other pixel of the block with alternating patterns for motion vector determination. Thus, a computation reduction of 4 is achieved while preserving the motion-estimation accuracy.

Figure 1.10a shows an example of alternating patterns used on a 4 × 4 pixel block. The motion vector of macroblock 3 is determined by group of pixels a, macroblock 2 uses group of pixels b, and so on. PD is reported to degrade the performance by approximately 0.1 dB compared to FBMA.

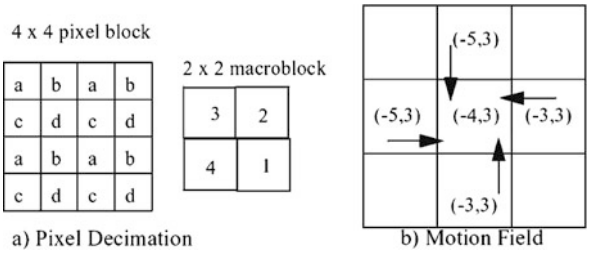

Fig. 1.10 Pixel decimation and motion field

A collection of all motion vectors defines a motion field. Motion fields of image sequences are usually smooth and slowly varying, with discontinuities limited to the boundary of objects moving in different directions or with different velocities. As a result, it is not uncommon to find neighbouring blocks with identical or near identical motion vectors. Motion estimation using motion field is performed in two steps. First, every other macroblock is estimated using any block matching algorithm. Then, the motion field is appropriately interpolated so that the motion vector of a block is determined from the predetermined motion vectors of the surrounding macroblocks (Fig. 1.10b). Interpolation requires only a small number of operations, so a sub-sampling of the motion field by a factor of 2 reduces the computational complexity by the same factor. The author also reported that sub-sampling the motion field by a factor of 2 causes only a minimal increase in prediction error [Liu93].

Later, Kim et al. [Kim95] extended PD to a 16:1 sub-sampling technique. This algorithm results in a degradation of less than 0.6 dB compared to PD.

1.4.3.4 Group 3: Reduction in the Bit-Depth of the Pixel

Belonging to the class of low-complexity BMA's, Gharavi et al. [Gharavi90] have proposed a technique where the pixels involved are classified into matched or

mismatched pixels, and hence the name Pixel Difference Classification (PDC). By definition, a pixel is matched when the absolute difference between the pixel in the reference block and the corresponding pixel in the candidate block is less than a threshold. The candidate block with the largest number of matching pixels is chosen to be the best match. The selection of matched/mismatched pixels primarily relies on a threshold which is subject to illumination changes within a frame or among frames. Feng et al. [Feng95] have introduced a Bit Plane Matching (BPM) technique, where the block mean is used as the threshold in contrast to an arbitrarily chosen threshold in PDC. If a pixel value is greater than the block mean, it is set to 1, otherwise, it is set to 0. Since block mean is required, there is a blocking effect on the generated binary frames, hence, the estimation performance is reduced by the block boundaries. In Natarajan et al.'s technique [Natarajan97], a 16 × 16 convolution kernel K which behaves like a band-pass filter, is applied, where:

$$K_{i,j} = \begin{cases} \frac{1}{25} & i,j \in [1,4,8,12,16] \\ 0 & \text{otherwise} \end{cases} \tag{1.36}$$

If a pixel in the original frame is greater or equal to the corresponding pixel in the filtered frame, the resulting binary pixel is set to 1; otherwise, it is set to 0. This technique, referred to as Band-Pass Bit Plane Matching (BP_BPM), is superior to PDC and BPM.

In fact, BP_BPM is a modification to BPM. While BPM calculates the thresholds using 256 values on fixed 16 × 16 windows, BP_BPM calculates the thresholds based on a 4-spacing-grid moving 16 × 16 windows. BP_BPM is, therefore, able to remove the blocking effect.

1.5 Statistical Redundancy Reduction Techniques

1.5.1 Run Length Coding (RCL)

The objective of image/video compression is to enable storage and transmission of visual data efficiently. In this case, the amount of visual data should be "compressed" in such a way that the final file size is optimally minimize while not compromising quality. A facsimile page can also be seen as an image composing of two levels of gray: black (represented by a "0") and white (represented by a "1"). If this page is scanned line-by-line, from the upper left of the page to the lower right, the result is a series of 0's and 1's. The total length of the series, or the number of source symbols, can be reduced by replacing pairs having a run of 0's and an ending 1, by pre-defined bit patterns. That is why *Run Length Coding* (RLC) was name for such technique.

RLC can be applied to series of zero-valued and non-zero transformed coefficients such as the Discrete Cosine transformed coefficients. The result of an n × n DCT, after quantization, is a series of n^2 coefficients, comprising many zeros. When

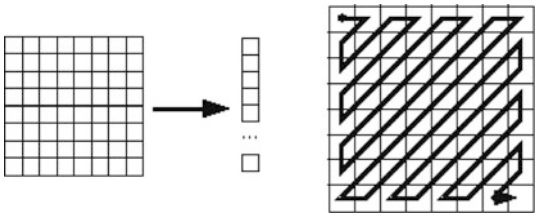

Fig. 1.11 An 8 × 8 transformed block is zig-zag scanned into a 1 × 64 vector with the largest values – in magnitude – concentrating at the beginning of the 1-D vector

the transformed coefficient is zig-zag scanned, the runs of zero-valued coefficients become apparent and a run length coder can be applied.

Recall from Sect. 1.3 on Spatial Redundancy Reduction Techniques that, the uncompressed 8 × 8 pixel block is discrete cosine transformed into an 8 × 8 pixel block of DCT coefficients. The so-called energy of the block is relocated to concentrate at the upper left corner of the block. In order to efficiently encode the transformed block, a zig-zag scanned sequence is developed as shown in Fig. 1.11.

Assuming the following zig-zag scanned sequence of 4 × 4 DCT coefficients is obtained:

$$55, 0, -32, 0, 0, 0, -5, 0, 0, 0, 0, 0, 2, 0, 0, 1$$

The first value, 55, is called a DC value (as opposed to AC values), and is coded differently. The outputs of the RLC are (1, –32), (3, –5), (5, 2), and (2, 1). It is noted that 16 coefficients have been reduced to 4 sets of values. Further reduction can be made using Huffman coding (Sect. 1.5.2), and Arithmetic coding (Sect. 1.5.3).

1.5.2 Huffman Coding

The sets of values (run-of-zeros, non-zero coefficients) can be further compressed using Huffman coding [Huffman52] as follows:

The more probable pair is coded with a shorter bit pattern, and all the bit patterns must be uniquely decodable.

The results of Huffman coding of the 4 pairs of values are shown in Table 1.1.

Table 1.1 Huffman codes for 4 pairs: (1, –32), (3, –5), (5, 2), and (2, 1)

Pair of values	Seen as symbols	Assumed freq. of occurrence	Assigned codeword
(1, –32)	A	0.500	0
(3, –5)	B	0.250	01
(5, 2)	C	0.125	011
(2, 1)	D	0.125	111

In Table 1.1, the pairs are arranged in decreasing order of their probability of occurrence. A look-up table (LUT) is established and as results, (1, –32) is represented by 0; (3, –5) by 01; (5, 2) by 011; and (2, 1) by 111. The resulting Huffman encoded bit-stream is 0, 0, 1, 0, 1, 1, 1, 1, 1. Of course, when decoding, the same table look-up must be used to translate 0 to (1, –32) and subsequently 0, –32; 01 to (3, –5) and subsequently 0, 0, 0, –5; 011 to (5, 2) and subsequently 0, 0, 0, 0, 0, 2; and finally 111 to (2, 1) and subsequently 0, 0, 1.

In case of no compression, a byte is used to represent each number in this sequence: 0, –32, 0, 0, 0, –5, 0, 0, 0, 0, 0, 2, 0, 0, 1. Thus, $8 \times 15 = 120$ bits would be needed. On the other hand, if RLC is applied, followed by Huffman coding, only 8 bits are needed to represent this same sequence. Therefore, a compression ratio of 120:8 or 15:1 is achieved.

1.5.3 Arithmetic Coding

Compared to the other lossless VLC techniques [Moffat02] including Elias, Golomb and Rice, Shannon-Fano, and Huffman [Huffman52] where each codeword must be represented by integer number of bits, codeword in Arithmetic Coding can be represented using *fractional number of bits*. Shannon first mentioned the possibility of such coding technique in 1948 [Shannon48]. Elias explores the idea of successive subdivision of coding interval [Abramson63] in the 1960s. The complete scheme of arithmetic coding was proposed by Rissanen [Rissanen76] and Pasco [Pasco76] independently in 1976, in which finite-precision arithmetic coding was implemented. Further research works include hardware-oriented arithmetic coders [Langdon84] of IBM and software-oriented arithmetic coders by Witten et al. [Witten87], which made it practical for applications requiring image and video compression.

The arithmetic coder generates codeword by representation of subintervals of the interval [0, 1] with fractional number of bits. The ratio of each subinterval to the current interval is proportional to the probability of occurrence of the corresponding coding event. Coding events can be of different types: pairs of (run of zeros, non-zero value), motion vectors, etc. Coding events of different types can be encoded using different LUTs.

If only two events are coded, 1 bit is used to represent the *most probable symbol* (MPS) and the *least probable symbol* (LPS). For this reason, it is called binary arithmetic coding and each coding symbol is called a bin.

As shown in Fig. 1.12, the coding interval for binary arithmetic can be defined as [Low, Low + Range), where Low indicates the lower boundary and Range indicates the range over which the probability of a coding event falls into. For each bin encoding, corresponding to MPS and LPS, the interval is subdivided into two sub-intervals: [Low_{MPS}, Low_{MPS}+ $\text{Range}_{\text{MPS}}$), and [$\text{Low}_{\text{LPS}}$, Low_{LPS} + $\text{Range}_{\text{LPS}}$); and a particular sub-interval is selected based on the probability of a coding bin. In this case, Low_{MPS} and Low_{LPS} represent the lower boundary of the sub-interval assigned to the MPS and LPS, respectively; while $\text{Range}_{\text{MPS}}$and $\text{Range}_{\text{LPS}}$ are

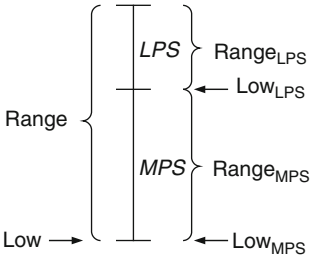

Fig. 1.12 Coding interval subdivision of binary arithmetic coding

the sub-intervals of MPS and LPS. Starting with the probability p_{LPS} of the LPS, $\mathrm{Range}_{\mathrm{LPS}}$, $\mathrm{Range}_{\mathrm{MPS}}$, $\mathrm{Low}_{\mathrm{LPS}}$, and $\mathrm{Low}_{\mathrm{MPS}}$ are calculated according to (1.37).

$$\begin{aligned} Range_{LPS} &= Range \times p_{LPS} \\ Range_{MPS} &= Range - Range_{LPS} \\ Low_{LPS} &= Low + Range_{MPS} \\ Low_{MPS} &= Low \end{aligned} \tag{1.37}$$

As coding interval is represented by finite number of bits, to overcome precision loss introduced by the shrinking of current interval and achieve incremental encoding and decoding procedure, an incremental output technique of arithmetic encoding is proposed in [Witten87], in which the interval is upscaled by left-shift of Range and Low when Range of interval is less then $1/4$ of the max range. This interval upscale procedure is named *renormalization*, during which, higher bits of Low need to be output as coding results. One bit of Low is output only when it is confirmed that the interval is within the upper half or lower half of the max interval range. Otherwise, the length of outstanding bits of Low is accumulated before the value of bits is confirmed. This coded bit output mechanism of binary arithmetic coding is adopted by CABAC of H.264/AVC.

1.6 Introduction to Video Coding Standards

Video coding technology has significantly changed the daily life of human beings in the last two decades. A variety of software/hardware applications of video coding technology have emerged recently. Because uncompressed video signals require huge amount of data storage and network bandwidth, video coding technologies are necessary to compress original video signals to reduce redundancy in spectral, spatial, temporal, and statistical domain. Several video coding standards have been established since 1980s to specify video coding techniques utilized for different applications, including H.261[ITU-T Rec. H.261], MPEG-1 [ISO/IEC 11172], MPEG-2 [ISO/IEC 13818-2], H.263 [ITU-T Rec. H.263], MPEG-4 Part 2 [ISO/IEC 14496-2], and H.264/AVC [ISO/IEC 14496-10].

H.261 [ITU-T Rec. H.261] is the first commercially viable video coding standard released in 1990. H.261 is established for low delay, slow motion applications

including video conference and video phone. It is transmitted over ISDN using small video frame size of CIF (352×288) or QCIF (176×144). This standard establishes the basic framework of block-based hybrid video codec (encoder and decoder) on which the subsequent standards are based on. In this block-based codec framework, each picture of video sequence is partitioned into macroblock (MB) of 16×16 pixels. The MBs are processed sequentially in intra or inter mode using several key techniques, include block-wise motion estimation and compensation for temporal redundancy reduction, discrete cosine transform (DCT) and quantization for spatial redundancy reduction, zig-zag scan of block coefficients and variable length coding (VLC) of different types of syntax elements (SEs) for codeword redundancy reduction, and low-pass in-loop filtering that helps to suppress the propagation of coding noise temporally.

In comparison to H.261, MPEG-1 [ISO/IEC 11172] introduces new technologies such as half-pixel motion estimation and bi-direction motion estimation (coding of B-pictures except I-picture and P-pictrure) to further reduce temporal redundancy. The MPEG-1 video sequence consists of GOPs (group of pictures), with I-pictures and P-pictures periodically inserted to enable random access of video sequence (decoding can start from any GOP). For I-picture coding, perceptual-based quantization matrix is used to more efficiently compress high frequency information of transformed coefficients, which is similar to that of JPEG [JPEG]. In H.261 and MPEG-1, high resolution and interlaced video coding is not supported.

Compared to earlier standards, MPEG-2 [ISO/IEC 13818-2] (also known as H.262) supports higher video quality and resolution at broadcast quality. It also supports both progressive and interlaced video formats targeting digital video broadcasting and DVD applications. More sophisticated motion estimation method, different DCT mode, and scanning method are developed in MPEG-2 to support interlaced sequence. It also provides various scalability modes including spatial, temporal, and SNR scalability. *Profile* and *level* are defined in the standard to allow applications to support only subsets of the features such as compression algorithm and chroma format. *Profile* defines the subset of coding tools that are used, while *level* defines the parameter range of a given profile.

H.263 [ITU-T Rec. H.263], which targets video conference over PSTN (Public Switched Telephone Network) and internet video, achieves a significant improvement of video compression, especially at low bit rate comparing to H.261. The updated versions of H.263 (H.263+ and H.263++) provide mode optional features that can further improve the efficiency of motion estimation and entropy coding. Compared to H.261, better motion estimation is achieved in H.263 because of half-pixel precision motion estimation, larger search range, and more efficient coding of motion vector (MV) by median prediction using neighboring SEs. For entropy coding of DCT coefficients, the 3D-VLC of (run, level, last) of H.263 is more efficient compared to 2D-VLC of (run, level) of H.261. Some of the optional techniques in the Annex of H.263 including variable block size motion estimation and arithmetic coding are adopted and further developed in the later standard H.264/AVC.

MPEG-4 Part 2 [ISO/IEC 14496-2] is the video coding standard developed by MPEG of ISO/IEC with improved quality compared to MPEG-2. To address various applications from low resolution low bit rate streaming video to high quality

DVD and HDTV broadcasting, 21 profiles are defined in MPEG-4 Part 2, and the most popular one deployed in the commercial codecs is Advanced Simple Profile (ASP). Quarter-pixel motion estimation is the new technique introduced in ASP that enhances the estimation accuracy, and it is later included in H.264/AVC. Some techniques of ASP are also criticized, such as the global motion compensation, which is not beneficial to compression and introduces negative impact of speed and implementation complexity. One important feature of MPEG-4 part 2: object-based video coding does not intend to improve video compression efficiency. Object can be represented by the shape, motion, and texture of the object separately. Because segmentation and tracking of object from video scene is a difficult task and requires high computation, object-based video coding is not efficient or practical in the real-time coding applications.

The latest video coding standard H.264/AVC (MPEG-4 Part 10) [ISO/IEC 14496-10] is developed targeting a wide range of applications and high compression capability. H.264/AVC was jointly developed by ITU-T and ISO/IEC, and gained rapid adoptions in a wide variety of applications, because of over 50% bitrate reduction achieved compared to the previous standards. Several profiles are defined in H.264/AVC, including Baseline, Main, Extended, High profiles, etc., with a set of technologies specified for each profile targeting a particular range of applications. H.264/AVC standard covers two layers: *Video Coding Layer* (VCL) that efficiently represents video contents, and *Network Abstraction Layer* (NAL) that formats the representation of VCL in the manner suitable for transport layer or storage media. Block-based hybrid video coding approach is utilized in VCL layer. A coded sequence of H.264/AVC consists of a sequence of *pictures*, and each picture is represented by either a *frame* or a *field*. Each picture is further partitioned into one or more *slices*, and each slice consists of a sequence of MBs. Slice is the smallest self-contained [Wiegand03] decoding unit in H.264/AVC bit stream. According to prediction modes, slices are commonly classified to 3 types, including *I slice* (intra prediction), *P slice* (single-direction inter prediction), and *B slice* (bi-direction inter prediction).

The block diagrams of MB encoding and decoding of VCL layer are shown in Fig. 1.13. As shown in Fig. 1.13a, MBs in each slice are sequentially processed at the encoder. Intra prediction or inter prediction including ME and motion compensation (MC) can be applied to each MB to reduce spatial or temporal redundancy. Integer transform and quantization are applied to reduce spatial redundancy by removing high-frequency information of prediction residual. Quantized residual data, intra/inter prediction data, and coding control signals are further compressed by the lossless entropy (statistical) coding to reduce redundancy of code words. An in-loop de-blocking filter is allocated in the MB encoding feedback loop to reduce artifacts at block edges and improve both subjective and objective visual qualities. The MB decoding procedure of H.264/AVC is illustrated in Fig. 1.13b, including entropy (statistical) decoding, inverse quantization and inverse transform, MC of intra prediction, and de-blocking filter. Computation complexity of decoding is significantly lower compared to encoding, because high complexity intra/inter prediction is not involved in decoding.

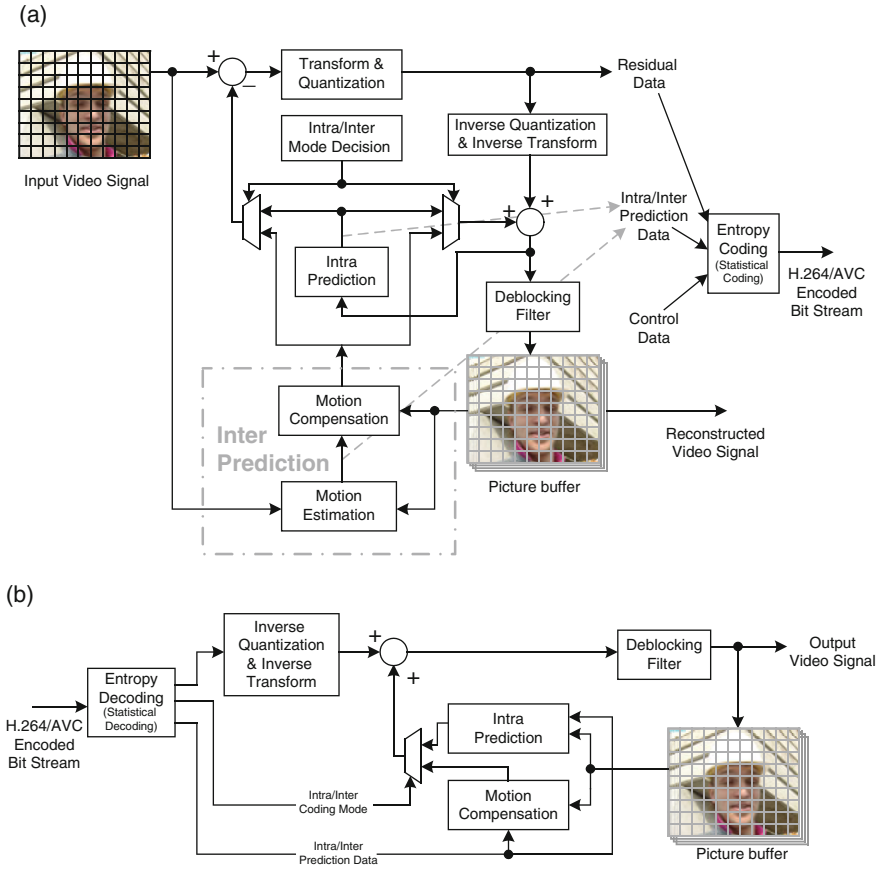

Fig. 1.13 Block diagram of MB processing in H.264/AVC. (**a**) MB encoding, (**b**) MB decoding

The significant improvement of compression efficiency of H.264/AVC is attributed to several techniques, including adaptive Intra16×16/Intra4×4 intra prediction, multi-reference ME and MC, and variable block-size and ¼-pixel precision of ME that reduce intra/inter prediction error, adaptive block-size (4×4 or 8×8) integer transform that efficiently concentrates energy of residual blocks with lower computation complexity compared to DCT, in-loop de-blocking filter that enhances both subjective and objective video quality, more efficient entropy coding tools including CAVLC [Bjøntegaard02] and CABAC [Marpe03a] compared to all previous standards, Rate-Distortion Optimization (RDO) [Sullivan98], etc. Moreover, adaptive frame/field coding at picture level (PAFF) and MB level (MBAFF) [Wiegand03, Yu06] is beneficial in some scenarios, compared to frame coding or field coding.

Intra prediction: The pixels of current processing block can be predicted pixels of neighboring coded blocks, and only prediction error and prediction mode are coded.

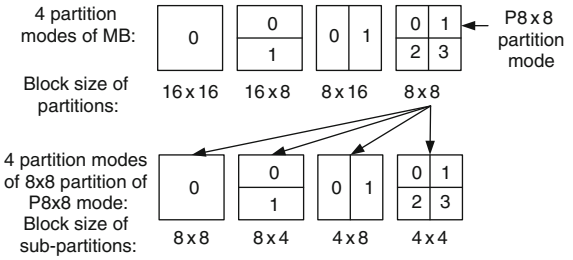

Fig. 1.14 MB partition modes and sub-MB partition modes of ME in H.264/AVC

The size of prediction block can be 16×16 or 4×4, and each intra-coded MB can be Intra16×16 or Intra4×4 based on prediction block size.

Inter prediction: Variable block-size ME of H.264/AVC supports more flexible selection of block size for motion compensation. As shown in Fig. 1.14, 4 types of MB partition modes P16×16, P16×8, P8×16, and P8×8 of ME are supported with corresponding partition sizes of 16×16, 8×16, 16×8, and 8×8 pixels. In addition, for mode P8×8, each sub-MB (8×8 partition) can be further partitioned into small partitions of 8×8, 8×4, 4×8, and 4×4 pixels. It enables better match of various motion patterns and more precise segmentations of motion regions, and contributes to the bit-rate reduction.

Rate-Distortion Optimization (RDO): RDO is applied to select best coding mode to achieve a minimum distortion D within a constrained bit-rate R. Lagrange multiplier methodology is applied to find minimum of RD cost (RD_{cost}) from all MB coding modes, as shown in (1.38). Multiplier λ is proportional to Quantization Parameter (QP). During MB encoding, RDO is applied to select MB or sub-MB partitioning mode of inter MB, Intra prediction mode of intra MB, best MV of ME, MB coding mode, etc. Coding rate R of RD_{cost} is precisely evaluated by entropy coding through CABAC or CAVLC.

$$\begin{aligned} RD_{\text{cost}} &= D + \lambda \cdot R \\ \lambda_{\text{mode}} &= 0.85 \cdot 2^{\frac{QP-12}{3}} \end{aligned} \tag{1.38}$$

Entropy coding: Two entropy (statistical) coding tools are utilized in H.264/AVC at the final stage of VCL including CAVLC (context-based adaptive variable length coding) [Bjøntegaard02] and CABAC (context-based adaptive binary arithmetic coding) [Marpe03a]. In the Baseline and Extended profiles targeting low bit-rate conversational network video service and stream services, CAVLC is utilized to encode the *syntax elements* (SE) of 4×4 block quantized transform coefficients, and Exp-Golomb coding is applied to encode other MB-level and high level SEs. For the Main and High profiles targeting high bit-rate and high definition service such as TV broadcasting or DVD, CABAC is used. CABAC achieves even higher compression ratio than CAVLC, with over 10% in bit-rate reduction. More details of arithmetic coding theory and CABAC will be introduced and analyzed in Chap. 2

Chapter 2
Review of CAVLC, Arithmetic Coding, and CABAC

2.1 Introduction of CAVLC

In the Baseline and Extended profiles of H.264/AVC, except the fixed-length coding, two VLC techniques are supported including: CAVLC for quantized transform residues and Exp-Golomb coding for other syntax elements [Wiegand03]. In the previous standards, entropy coding of residues is based on the forward zig-zag scanned run-length coding and fixed variable-length coding. The H.264/AVC standard adopts backward the zig-zag scanned run-length coding with adaptive VLC.

After intra/inter prediction and 4×4 transform and quantization of prediction residuals, CAVLC [Bjøntegaard02] is used to encode the quantized transformed coefficients, and *Universal Variable Length Coding* (UVLC) is used to encode other types of SEs.

The motivations of using CAVLC for SEs of transform coefficients are as follows: (1) Context adaptive coding can provide better coding efficiency over the whole quality range of the standard than UVLC technique; and (2) Separation of the coding of Run and Level, due to possession of weak correlation, after zig-zag scan can achieve better adaptivity, coding efficiency, and lower the demand on memory than that of Run-Level pair coding of UVLC.

After transformed and zig-zag scanned, the non-zero coefficients are often sequences of ±1. The CAVLC coder indicates the number of high frequency ±1 coefficients by using trailing 1s (*T1s*). For non-zero coefficients, coding efficiency can be improved by using different VLC tables.

An example of CAVLC encoding of one 4×4 block is illustrated in Fig. 2.1. After zig-zag scan, the reordered coefficients are coded as sequence of SEs, including *coeff_token*, *sign_T1s*, 2 values for *level*, *total_zeros*, and 4 values for *run_before*.

These 5 types of SEs will be discussed as follows.

1. *coeff_token* is used to code both the total number of non-zero coefficients (*total_coeff*) and the number of trailing ones (*T1*s). Note that the maximum T1s allowed is 3. Any extra 1-valued coefficient is treated as normal non-zero coefficient. In this case *coeff_token* = 0000100
2. *sign of T1*s is used to code the sign bit of each T1 in reverse zig-zag scanned order. *sign_of_T1* = 011

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6_2,

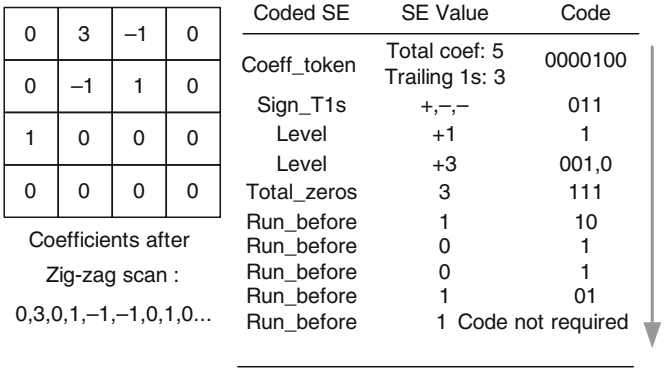

Fig. 2.1 CAVLC encoding of one transform coefficient block [Richardson03]

3. *level* (in sign and magnitude) is used to code the value of each of the remaining non-zero coefficients in reverse order, with the highest frequency non-zero coefficient being encoded first. *level*(1) = 1 (prefix); and *level*(2) = 001 (prefix), and 0 (suffix)
4. *total_zeros* is used to code the total number of zeros preceding the last non-zero coefficient. *total_zeros* is 3 and coded by "111"
5. *run_before* is used to code the number of successive zeros preceding each non-zero coefficient in reverse zig-zag scanned order. *run_before* is coded with 10, 1, 1, 01.

The resulting encoded bit-stream is 0000100_011_1_001_0_111_10_1_1_01.

As shown in Fig. 2.2, both CAVLC decoding and encoding must be executed in 6 steps to completely process the 5 types of SEs of each coefficient block.

From the implementation viewpoint, the *coeff_token* is often obtained by looking up in a 2-D LUT. The selection of LUT of luminance block is based on the number of non-zero coefficients of the previously coded blocks positioning on top and on the left of the block at issue. The *sign_T1s* are output with 1-bit/sign. The *Level* values of the remaining coefficients are coded using selected 1-D LUTs. Each Level-LUT

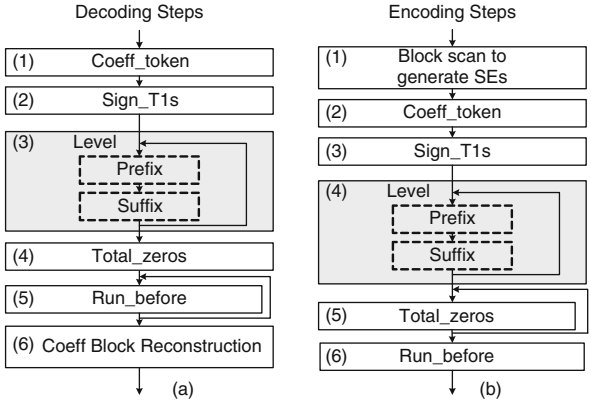

Fig. 2.2 Flowchart for CAVLC codec: (**a**) Decoder and (**b**) Encoder

consists of Exp-Golomb codewords [Golomb66] for levels with small values (associated with high probability), and escapes codewords for levels with large values (low probability). The *Run* values of the block are coded in two steps in LUT: First, the coding of *total_zeros* before the last non-zero coefficient, and second, the coding of *run_before* each non-zero coefficient in the reverse scanned direction.

The coding efficiency of the scheme is high, as the probability is high that one or more low frequency coefficients have no Runs, especially in high bit rate coding. Because the maximum of *total_zeros* is related to non-zero coefficient number, the 1D-LUTs of *total_zeros* are allocated separately for different coefficient number. For *run_before* coding, the number of zeros left to be coded is utilized to select LUT of *zeros_left*. Context adaptivity is introduced to CAVLC coding of all SEs, except *sign_T1s*.

2.2 Pre-CABAC Arithmetic Coders

Compared to the previously reported lossless variable length coders (VLC) [Moffat02] including Elias, Golomb and Rice, Shannon-Fano, and Huffman [Huffman52], the distinct difference of arithmetic coding is that codewords can be represented using fractional number of bits, while in other VLCs, each codeword must occupy integer number of bits. Shannon first mentioned the possibility of such coding technique in 1948 [Shannon48]. Elias explores the idea of successive subdivision of coding interval [Abramson63] in 1960s. Complete scheme of arithmetic coding was proposed by Rissanen [Rissanen76] and Pasco [Pasco76] independently in 1976, in which finite-precision arithmetic coding was implemented. Further research works include hardware-oriented arithmetic coders [Langdon84] of IBM, and software-oriented arithmetic coders by Witten et al.[Witten87], which made it practical in the image and video compression applications.

Multiplication of $\text{Range}_{\text{LPS}}$ (1.38) of subdivision of arithmetic requires high computation and limits the arithmetic coding throughput. Before CABAC, three types of multiplication-free binary arithmetic coders have already been proposed for image compression, which significantly reduce computation complexity of arithmetic coding. The pre-CABAC arithmetic coders include Q-coder [Pennebaker88], and its variants QM coder [Mitchell93] and MQ coder [Taubman00, Taubman02]. QM coder is adopted as the entropy coder in *Joint Bi-level Image Experts Group* (JBIG) standard [JBIG] and *Joint Image Experts Group* (JPEG) standard [JPEG], while MQ coder is adopted in JPEG2000 standard [JPEG2000]. The Q-coder, QM coder, and MQ coder are reviewed as follows.

2.2.1 Q-Coder

Q-coder is an adaptive binary arithmetic coder suitable for both HW and SW implementations. It has simple adapting mechanism which operates on the estimated

probability of coding bin during the coding process. Q-coder avoids the increasing-precision problem using fixed-precision [Rissanen76]. Renormalization is applied after interval subdivision to maintain the size and precision of coding interval. Carry propagation during renormalization is solved by "bit stuffing". When 8 successive bits of "1"s are output, a "0" is stuffed as a carry gap. Multiplication is avoided by using approximated value 1 of Range, because Range is in [0.75, 1.5). Probability estimation is only taken when renormalization is needed for the encoding bin. Computation reduction of interval subdivision and renormalization helps accelerate encoding. However, the precision of arithmetic coding is sacrificed and compression efficiency is degraded. The probability estimation is based on state transition of *Finite-state machine* (FSM), which can be implemented as simple table lookup operation.

2.2.2 QM Coder

QM coder enhances the coding efficiency of Q-coder by incorporation of conditional exchange, and state machine for probability estimation using Bayesian estimation principle for rapid initial learning. The differences between QM coder and Q-coder include [Slattery98]: (a) MQ coder has higher precision with 3 more bits allocated for the coding interval and other intermediate parameters; (b) Q-coder is hardware-based, and QM coder is software-based; (c) Carry propagation is solved at decoder of Q-coder, while it is solved at encoder of QM coder. Bit stuffing is used in Q-coder to prevent carry propagating in the encoder, which is faster than byte-stuffing of QM coder. However, the carry still needs to be solved in the decoder of Q-coder. In comparison, QM coder resolves the carry propagation at encoder by byte stuffing technique, and processing of carry is not needed at decoder.

In the QM coder, adaptive probability estimation is implemented by estimation of next $\text{Range}_{\text{LPS}}$ when renormalization is taken for the coding bin. The estimation is performed by a table lookup operation, and the values of MPS and LPS can be exchanged when the probability of LPS reaches 0.5. For binary arithmetic coding of QM coder, bit assignment of Low and Range is shown in Fig. 2.3. Low register consists of one carry bit, 8 bits of output byte, 3 bits of spacer, and 16 bits of active region. Range register also has 16 bits of active region. During renormalization process, a counter counts the number of valid bits in the output byte section of Low register. When the output byte is full, the byte can be output or recorded as an outstanding byte based on carry bit and whether the output byte is 0xFF. Carry bit of

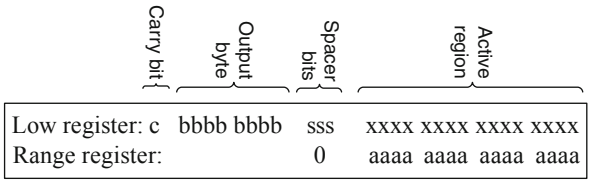

Fig. 2.3 Bit assignment of low and range in QM coder of JPEG

Low can only influence the last byte of bytes in the output buffer. Three spacer bits are allocated in Low between output byte and active region to reduce probability of carry overflow to the output buffer.

2.2.3 MQ Coder

In JPEG2000 [JPEG2000, Taubman02], MQ coder is utilized as the entropy coder of the *Embedded Block Coding with Optimized Truncation* (EBCOT) to encode bit planes of code block of quantized image subbands after *Discrete Wavelet Transform* (DWT) and quantization. MQ coder is closely related to QM coder of JPEG. One distinct difference between the two coders is that the carry is fully resolved in QM coder of JPEG indicating a carry in the Low register will propagate to the nearest bit 0 in the output bits. In comparison, carry propagation in MQ coder is restricted by the mechanism that one bit of 0 is stuffed when the output byte is 0xFF. Any carry from the following renormalization operations cannot influence previous output bytes. Compared to the renormalization and output bit packing mechanism of Q-coder and QM coder, bit stuffing of MQ coder is more efficient in bandwidth and processing time as only one redundant bit stuffing is needed when byte 0xFF is output, and the counter for outstanding bytes is not necessary. ASIC design [Pastuszak05] is reported to accelerate the coding throughput of MQ coder and EBCOT.

2.3 CABAC of H.264/AVC

Although Q-coder, MQ coder, and QM coder are also binary arithmetic coders with statistics adaptivity features, the *Context-based Adaptive Binary Arithmetic Coder* (CABAC) – proposed by Marpe et al. in 2001 [Marpe01a, Marpe01b, Marpe01c] – is adopted as the entropy coding tool for the Main profile and higher profiles of the H.264/AVC standard. Before CABAC, LUT-based VLC are generally utilized for the entropy coding stage in the hybrid block-based video coding standards including H.263, MPEG-2, MPEG-4 Part 2. The weakness of VLCs is that coding event with probability larger than 0.5 cannot be efficiently represented, and the coding procedure is not adaptive to the actual symbol statistics as the values of LUTs are fixed [Marpe03a]. The only arithmetic coder adopted in video standard is of Annex E of H.263 [ITU-T Rec. H.263], in which coding efficiency of entropy coding is not significantly improved, because it directly uses SEs of VLC for arithmetic coding without redefinition. Before CABAC proposals by [Marpe01a, Marpe01b, Marpe01c] of H.264/AVC, similar arithmetic-coder-based implementations were first investigated and applied to the non-block-based video coding [Marpe99, Heising01], such as DWT.

CABAC [ISO/IEC 14496-10, Marpe03a] is the first successful arithmetic coding scheme deployed in the video coding standard, with significant compression improvement compared to the previous entropy coding tools. As shown in Fig. 2.4,

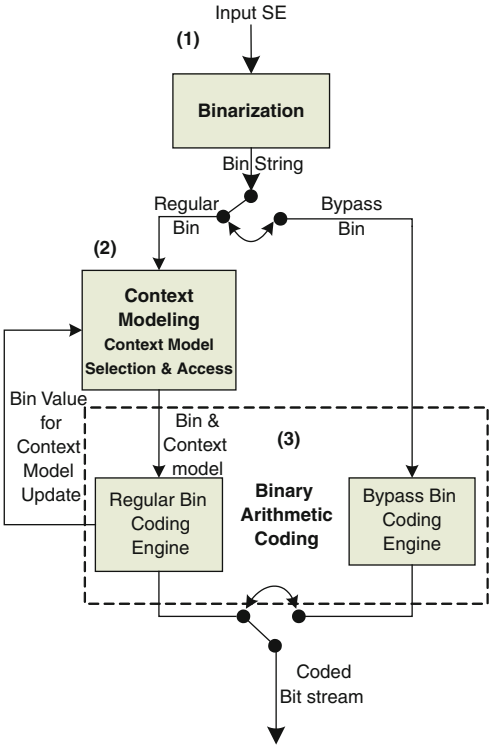

Fig. 2.4 Block diagram of the CABAC encoder of H.264/AVC

CABAC encoding process consists of three elementary *steps*: binarization, context modeling, and binary arithmetic coding (BAC). The input SEs are binarized into bin strings, in which regular bins and bypass bins are encoded separately by the encoding engines of BAC. For regular bin coding, the context model (probability model) of the bin is prepared by the step of context modeling. Details of the three steps are discussed in the following sections.

2.3.1 Binarization

Binarization maps the non-binary valued SE into bin string, which is a sequence of binary decision (bin). Three types of bins are generated in the binarization step: *regular bin*, *bypass bin*, and *terminate bin* for the bins with unequal (variable), equal, or dominant probabilities of value 1 and 0, respectively. Advantages of binarization [Marpe03a] include: (a) the probability of non-binary SE can be represented by the probabilities of individual coding bins, while compression efficiency is not influenced; (b) low-complexity BAC can be utilized; (c) context modeling at sub-symbol (sub-SE) level provides more accurate probability estimation than context modeling at the symbol level, and the alphabet of the encoder is reduced.

Five binarization schemes are used in CABAC: Unary (U), Truncated Unary (TU), kth order Exp-Golomb (EGk), concatenation of the first and third scheme (UEGk), and fixed length binarization (FL). Kth ordered Exp-Golomb binarization (EGk) [Teuhola78] – a derivative of Golomb coding [Golomb66] – is proved to be an optimal prefix-free coding for geometrically distributed sources. EGk codeword consists of prefix and suffix bin strings, with total length of $2l+k+1$ bits. EGk prefix is a Unary codeword, with l bits of 1 and one terminating bit 0. The length l of string of bit 1 is represented as:

$$l = \left\lceil \log_2 \left(\frac{x}{2^k} + 1 \right) \right\rceil \tag{2.1}$$

The length of suffix binary string is equal to $l + k$, and the value of the suffix string is:

$$EGk_{suffix} = x + 2^k - 2^{k+l} \tag{2.2}$$

UEGk is combined binarization scheme of TU and EGk. It is utilized for binarization of SEs of absolute value of residual coefficient level and MVD. TU generates prefix of the bin string, and EGk is adopted to generate the suffix with k set to 0 and 3 for coefficient level and MVD, respectively. TU is simple and it permits fast adaptation of probability of coding symbol. However, it is only beneficial for small SE values. For large SE values, suffix bin string generated by EGk provides a good fit to the probability distribution, and bypass bin coding is also utilized to reduce computation complexity.

2.3.2 Context Modeling

The context modeling step – shown in Fig. 2.4 – implements two sub-functions: context model selection and context model access. The statistics of the coded SEs are utilized to update the probability models (context model) [Wiegand03] of regular bins. For regular bin coding, one context model is chosen, and fetched from a pre-defined set of context models to provide the probability of regular bin to be MPS or LPS, and the context model is updated after bin coding based on bin value. The context index (*CtxIdx*) is calculated to select context model, which is the sum of context offset (*CtxOffset*) and context index increment (*CtxIdxInc*). *CtxOffset* locates the context model set of processed SE, while *CtxIdxInc* selects one context model from the set based on the values of coded bins or coded SEs of neighboring coded blocks.

The idea of multiplication-free arithmetic coding of H.264/AVC is based on the assumption that estimated probability of each context model can be represented by a sufficiently limited set of representative values. In CABAC, the number of the representative values is set to 64 to enable accurate estimation, which is larger than the 30 of Q-coder. Each context model contains a 1-bit tag of MPS value, and a 6-bit probability state index (*pStateIdx*) that addresses one of 64 representative probability

values of LPS from p_0 to p_{63} in the range of [0.01875, 0.5]. The probability values of LPS are derived from (2.3). The ratio of two neighboring probability values is a constant value α, which is approximately equal to 0.949.

$$\begin{aligned} &p_\sigma = \alpha \cdot p_{\sigma-1} \\ &for: \sigma = 1, \ldots, 63, \quad \alpha = \left(\frac{0.01875}{0.5}\right)^{1/63}, and \quad p_0 = 0.5 \end{aligned} \tag{2.3}$$

Probability update of context model is based on the rule in (2.4), in which p_{old} and p_{new} are the probabilities for the bin to be LPS before and after bin coding. If the coding bin is MPS, the probability of LPS decreases by simply multiplying the ratio α, while for the LPS bin, the update probability of MPS is calculated first, and then the probability of LPS is obtained.

$$p_{new} = \begin{cases} \max(\alpha \cdot p_{old}, p_{62}), & if \quad bin = MPS \\ 1 - \alpha \cdot (1 - p_{old}), & if \quad bin = LPS \end{cases} \tag{2.4}$$

By mapping the update probability value of LPS of (2.4) to the closest value in the aforementioned set of representative values, multiplication of probability estimation of CABAC is replaced by simple table lookup for the *pStateIdx* of next probability state according to *pStateIdx* of the current bin, and based on whether it is MPS or LPS. This probability value estimation of context state is actually the function state transition of FSM with 64 predefined states. This type of probability FSM is first utilized in Q-coder, and adopted in QM coder and MQ coder. Compared to Q-coder, QM coder, and MQ coder, the representative LPS probability values need not to be stored in CABAC. Instead, the approximation of the products of coding interval Range and the LPS probability of (1.37) are stored. In order to be more adaptive to the coding context, the values of MPS and LPS can be exchanged when the probabilities of MPS and LPS are equal and the coding bin is LPS.

For particular regular bins of CABAC, multiple context models are allocated for single bin to more precisely represent probabilities of bin in different coding contexts. Four types of context model selection techniques are supported in CABAC, based on (a) neighboring coded SE values of the current SE, (b) values of prior coded bins of SE bin string, (c) position of the to-be-encoded residual coefficient in the scanning path of residual block coefficients, and (d) level values of encoded coefficients of residual block.

2.3.3 Binary Arithmetic Coding (BAC)

BAC performs arithmetic coding of each bin based on bin value, type, and the corresponding context model of the bin. BAC is a recursive procedure of coding interval subdivision and selection, as shown in Fig. 2.5.

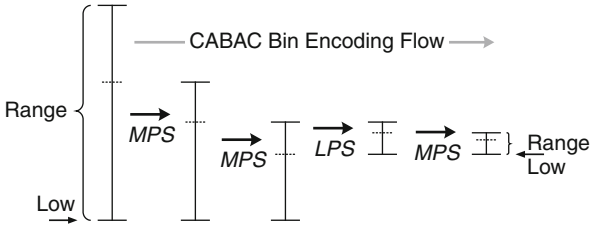

Fig. 2.5 Coding interval subdivision and selection procedure of CABAC

Coding interval subdivision mechanism of CABAC is different from that of QM and MQ coders. In QM and MQ coders, calculation of $\text{Range}_{\text{LPS}}$ of (1.38) is simplified by using the approximated value 1 for Range, and multiplication is eliminated. In comparison, Range is also utilized for $\text{Range}_{\text{LPS}}$ calculation of CABAC. Figure 2.6 shows the reference pseudo C-program of interval subdivision and selection of regular bin, in which the 2 higher-order bits of Range (bit 7 and bit 6) and the index of probability state (*pStateIdx*) of LPS are used to look up for the precalculated product of Range and p_{LPS} of (1.37) in a 2-D LUT. Although the product of LUT is of limited precision, the precision of $\text{Range}_{\text{LPS}}$ calculation, interval subdivision is improved, and computational complexity is reduced in CABAC, compared to those of QM and MQ coders.

```
RangeIdx =  (Range >> 6) & 3; //Range[7:6]
Range_LPS = rangeTableLPS[pStateIdx][RangeIdx];
Range_MPS = Range - Range_LPS;
if(bin == MPS) //bin is MPS
  Range = Range_MPS;
else { //bin is LPS
  Range = Range_LPS;
  Low = Low + Range_MPS;
}
```

Fig. 2.6 Coding interval subdivision and selection of regular bin of CABAC

Because Range and Low of coding interval are represented by finite number of bits (9 bits for Range and 10 bits for Low), it is necessary to renormalize (scale up) the interval to prevent precision degradation, and the upper bits of Low are output as coded bits during renormalization. Coding interval renormalization and bit output of CABAC is based on Witten's algorithm [Witten87], as illustrated in the reference pseudo C-program of Fig. 2.7. The coding interval of (Low, Low + Range) is renormalized when Range is smaller than the threshold value 256 (0x100), which is ¼ of the maximum range of coding interval.

As illustrated in Fig. 2.7, renormalization of Range and Low is an iterative procedure, and the maximum number of iterations is 6, as the smallest possible value of Range is 6. For the processing of carry propagation and output of coding bits, the coded bits of CABAC are not output until it is confirmed that further carry

```
while (Range < 0x100) {
  if (Low >= 0x100) {
     if (Low >= 0x200) {
       //Output bit 0 and following outstanding bits of 1
       PutBit(1);
       Low = Low - 0x200;
     }
     else {  // Low is between 0x100 and 0x200
       Low = Low - 0x100;
       NumOutstandingBits ++;  //Accumulate outstanding bits
    }
  }
  else {
    //Output bit 0 and following outstanding bits of 1
     PutBit(0);
  }
  //scale up Range and Low values by left shift
       Range = Range << 1;
       Low = Low << 1;
}
```

Fig. 2.7 Pseudo C-program of renormalization and bit output of CABAC

propagation will not influence bit values. Figure 2.7 illustrates that only when interval length (Range) is smaller than the threshold 0x100, one bit can be output if the interval is located within the top half [0x200, 0x400) or bottom half [0, 0x200) of the maximum coding range, or an outstanding (OS) bit is accumulated when the interval is within [0x100, 0x300). When a bit of value *X* is output in BAC, the accumulated OS bits are output with the value 1–*X*. Compared to the bit stuffing or byte stuffing schemes of Q-coder, QM coder, and MQ coder, carry propagation is completely solved during renormalization of BAC, and no additional processing of bit stream is needed at CABAC decoder. Moreover, as no bits or bytes are stuffed in the bit stream, the compression efficiency of CABAC is further improved. However, the renormalization illustrated in Fig. 2.8 is a highly sequential operation, and as the number of iterations is variable depending on the selected subinterval Range, it is

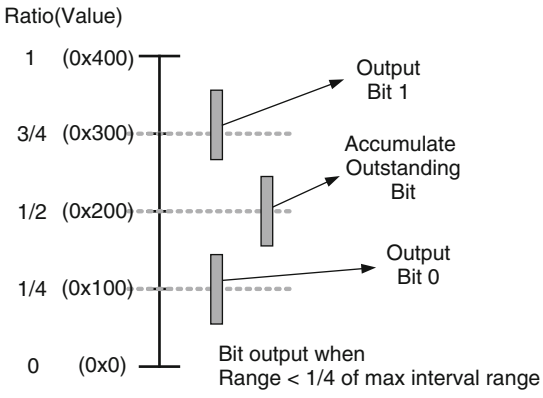

Fig. 2.8 Decision of bit output and accumulation of outstanding (OS)bit

challenging for SW or HW acceleration of renormalization and bit output of BAC. In some situations, long delay may be experienced, when a large number of OS bits are accumulated.

2.3.4 Comparisons of CABAC with Other Entropy Coders

The coding efficiency of CABAC is higher than those of the Q-coder, QM coder, and MQ coder, because of (a) the more precise multiplication of $\text{Range}_{\text{LPS}}$, (b) larger number of probability states for each probability model, and more precise probability estimation of coding bins; and (c) more context models (probability models) deployed for various coding contexts of different types of SEs.

Because of the high computational complexity of CABAC, another entropy coding tool CAVLC [Bjøntegaard02] is deployed in the Baseline profile and Extended profile of H.264/AVC targeting low bit-rate real-time video coding. It offers compression-complexity tradeoff with lower complexity, and lower coding efficiency, compared to CABAC [Marpe03a]. It is employed to encode the quantized transform coefficients of 4×4 residual blocks, while zero-order Exp-Golomb codes [Teuhola78] (EG0) are used for all other types of non-residual SEs. Adaptivity is introduced to CAVLC by allowing switching among multiple VLC tables based on the already processed SEs, and the coding efficiency of CAVLC is better than those of the previous VLC coders which used single VLC table. Instead of coding data pair of run-level as a single SE, run and level of the residual block are encoded separately in CAVLC, so that the inter-symbol redundancy can be more efficiently exploited. However, compression efficiency of CABAC is significantly higher, with typically bit rate reduction of 9–14% in the video quality range of 30–38 dB [Marpe03a], compared to CAVLC and EG0. This is because (a) in CABAC, encoding symbols can be more precisely represented in non-integer number of bits, especially for the symbol with probability higher than 0.5, and (b) CABAC encoder is more adaptive to the non-stationary symbol statistics with efficient context modeling (probability estimation) for the coding bins of all types of SEs.

Since the adoption of CABAC entropy coding in H.264/AVC [Marpe97, Marpe, Heising01, Marpe03a, Mrak03], CABAC is also applied in many applications of image and video processing including motion mode and residual data of 3D dynamic mesh [Muller05], prediction residual in lossless 4D medical image compression [Sanchez08], SEs of 8×8 transform coefficients of AVS coding standard [Zhang07], motion vector coding of scalable video coder [Wu07], parameters of depth and correction vectors in multi-view video coding [Sehoon07]. CABAC is also utilized to encode affine motion vector [Kordasiewicz07], and MVD of 3-D DWT-based subband video encoder [Golwelkar07].

Chapter 3
Review of Existing Statistical Codec Designs

3.1 Acceleration Approaches for H.264/AVC Codec

The computational complexity of H.264/AVC is known to significantly higher than to those of the previous standards, and quite a number of researches have been carried out to accelerate the H.264/AVC encoding or decoding using algorithm improvement, SW optimization, and HW acceleration both at the system level and at particular functional blocks.

Algorithm improvement is carried out in the aspects of enhancing accuracy of the context model selection in MVD coding [Sun07], investigating parallel CABAC coding using table lookup technique with parallelized probability models [Lin06b], analyzing error detection probability (EDP) of CABAC coded SEs [Levine07], error resilience enhancement of coded bit stream by inserting detective markers based on CABAC semantics [Li06c], or error detection based on joint source-channel MAP estimation [Wang05, Jamaa06].

For SW optimization, DSP-based H.264/AVC encoder designs are reported in [Lahti05, Kant06, Lin06a, Wei07], while Cell processor-based [Baik07] and ARM processor-based [Huang08] implementations are reported to achieve low-resolution SW decoding. Fast algorithms are developed to accelerate particular function blocks such as intra prediction [Pan04], coding mode decision and RDO [Kim04, Ba06, Nieto06, Yu06], ME and MC [Li07b], and rate control [Ma05, Schwarz07]. However, SW optimization is limited by the low level of parallelism, and is not suitable for high bit-rate high definition real-time coding.

Hardware acceleration of H.264/AVC codec is reported in large number of literatures targeting encoder/decoder system or particular function blocks. For encoder design, MB encoding is accelerated by 4-stage pipeline [Huang05, Chen06a] or 3-stage pipeline [Liu07b] to enable parallel processing of different MB coding steps such as integer ME, fractional ME, transform and quantization. To remove data dependency and enable pipelined coding, the algorithm is adjusted, including simplified MV prediction in [Huang05, Chen06a]. Different encoding stages are controlled by the embedded processor [Inata08] or through control signals

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6_3,

input from the system bus interface [Liu07b]. As the computation complexity of the decoder is significantly lower, FPGA is utilized to achieve real-time decoding excluding entropy decoding in [Agostini06]. Schemes of memory access reduction and memory size reduction of decoder are reported with strategies of optimized scheduling of decoding order [Chen06b], data reuse by allocation of shared memory and local buffers [Huang05, Chen06a, Liu07b, Lee08], and multi-bank SRAM access [Chang07b]. Power reduction and chip testing schemes of codec are considered in [Lin08b]. HW designs that focus on accelerating of particular functional block have also been reported. To reduce the ME computational complexity, The MB partition modes and the number of search candidates are reduced in [Liu07b], full-search early termination of ME is applied in [Lee08], and control of search range and reference frame number in [Murachi08] according to input variations. However, video quality is also degraded [Babionitakis06] with such simplification. Hardware SIMD architecture of ME is designed in [Sayed06] to take advantage of the level of parallelism possessed in the algorithm. For MC at the decoder, interpolation window reuse scheme [Tsai05b] is utilized to reduce memory bandwidth. For intra prediction, acceleration strategies are proposed including prediction mode decision with reference to the mode of coded blocks [Chang07a] and scheduling of parallel processing of Intra16×16 and Intra4×4 prediction [Tsai05a].

Recognizing the highly computational complexity of motion estimation, because of the serial nature and high data dependency of the coding procedure in both CAVLC and CABAC, the throughput of a H.264/AVC video codec is also limited by the entropy coding stage. As it is not efficient to remove the bottleneck by algorithm simplification and software optimization alone, a number of hardware designs for CAVLC and CABAC have been proposed in order to (i) enhance the throughput; (ii) reduce the area; (iii) reduce the power consumption in various applications.

In the following sections, the CAVLC decoder and encoder designs will be reviewed first followed by the review of CABAC decoder and encoder designs. Analyses on implementation strategies will be performed based on performances such as throughput (in SE/cycle in the case of CAVLC, and bin/cycle in the case of CABAC), coding speed (in SE/second for CAVLC, and bin/second for CABAC). The costs of the designs will be analyzed in terms of circuit area (in both logic gates and memory bits), as well as power consumption (in mW, if data available).

3.2 CAVLC Decoder and Encoder Designs

CAVLC has its name from two major concepts: variable-length coding and time-varying input statistics adaptivity. The variable-length coding involves run-length coding followed by a mapping of symbols into codewords of variable lengths. Generic implementations of high performance designs include parallelism/pipelining of different coding steps, table lookup acceleration, and processing of multiple symbols in a single cycle.

3.2.1 CAVLC Decoder Designs

According to the H.264/AVC standard, associated with CAVLC, there are 5 SEs: *coeff_token*, *sign_T1*, *level*, *total_zeros*, and *run_before*. *Sign_T1* and *levels* are decoded using simple arithmetic operations since these two SEs are coded by regular VLC codes. For the other SEs, *coeff_token*, *total_zeros*, and *run_before* are coded by the content-dependent VLC codes, and thus decoding is performed by LUTs. CAVLC decoder can be realized in 6 steps:

S1. Parse the coeff_token from the input bit-stream by a LUT, and obtain the number of non-zero coefficients and trailing ones (T1s).
S2. Parse the sign_T1s to get the signs of trailing 1s, and reconstruct the values of trailing 1s in the residual block.
S3. Parse the prefix and suffix of level to calculate level values, collect, and buffer all level values.
S4. Parse the total_zeros.
S5. Recursively parse run_before values of the block, and
S6. Reconstruct all coefficients of the block according to run-level data pairs in the reverse zig-zag scanned order.

In the following sections, the reported CAVLD designs will be reviews in their differentiating features and performances.

3.2.1.1 [Chang05]

Chang et al. [Chang05] proposed a CAVLD which includes: a PCCF (Partial Combinational Component Freezing) feature, a HLLT (Hierarchical Logic for Look-up Tables), a ZTEBA (Zero-left table elimination by arithmetic), a IDS (Interleaved Double Stacks), and a ZCS (Zero Codeword Skip). Figure 3.1 shows the CAVLD architecture.

This design can be seen as one of the most direct implementations of the CAVLD algorithm. As shown in Fig. 3.1, the functional blocks are implemented one-by-one with a global feature to enable/disable individual block (PCCF – Partial Combinational Component Freezing) to supposedly save power. Large LUT with unused memory space is divided into smaller LUTs, and through a controller, search is logically performed in a hierarchical manner (HLLT – Hierarchical Logic for Look-up Tables) from the LUT with the most probable codewords to the LUT with the least probable codewords. An IDS (Interleaved Double Stacks buffer) helps buffer the data traffic back and forth between the CAVLD and the Inverse Quantizer.

It was claimed in [Chang05] that the decoding steps S1 and S2 are performed in one cycle. However, because these two steps are part of a multiple-step process, the average decoding cycles of each block is not necessarily reduced.

This design takes up 9.9 Kgates and 1.1 Kbits of memory using 0.18 μm CMOS, and can operate at 175 MHz for decoding HD1080i video, and yields a throughput of 1 SE/cycle. No figure on power consumption reported.

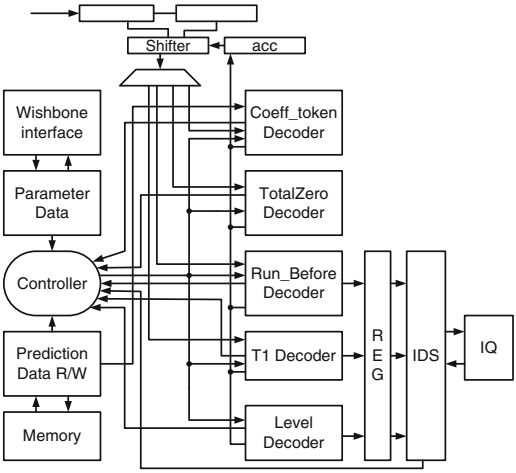

Fig. 3.1 CAVLD architecture [Chang05] with direct implementations of functional blocks

3.2.1.2 [Alle06]

In [Alle06], decoding is performed in 5 different stages corresponding to the 5 types of SEs: *coeff_token*, *sign _T1*, *level*, *total_zeros*, and *run_before*, as shown in Fig. 3.2. It is claimed that "decoding more than one SE in a clock cycle would decrease the clock operating frequency". Thus, the aim is to process one SE in one cycle. The original LUTs can be divided into sub-tables based on the number of leading zeros (LZ) in the codewords. As shown in Fig. 3.3, sub-table selection is implemented by a fast LZ detection (LZD) circuit.

Parsing of *sign_T1* can be implemented in a single cycle, as it is fixed-length SE parsing, and the number of T1s is already obtained in S1. In S5, because the codeword of *run_before* LUT is short (3 bits or less), 2 values of *run_before* are decoded in a cycle in [Alle06] with minimum addition to circuit area. When the *run_before*

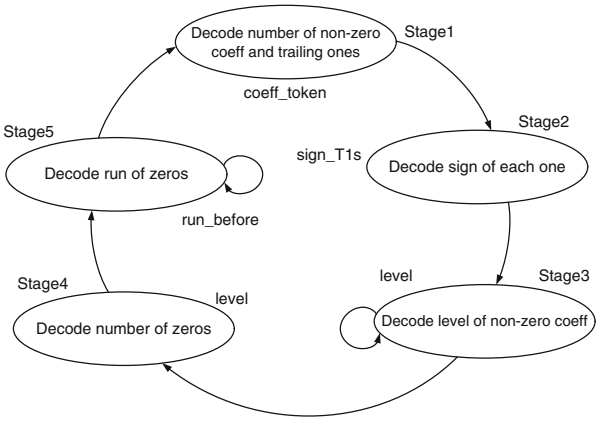

Fig. 3.2 Five decoding stages of the CAVLD architecture [Alle06]

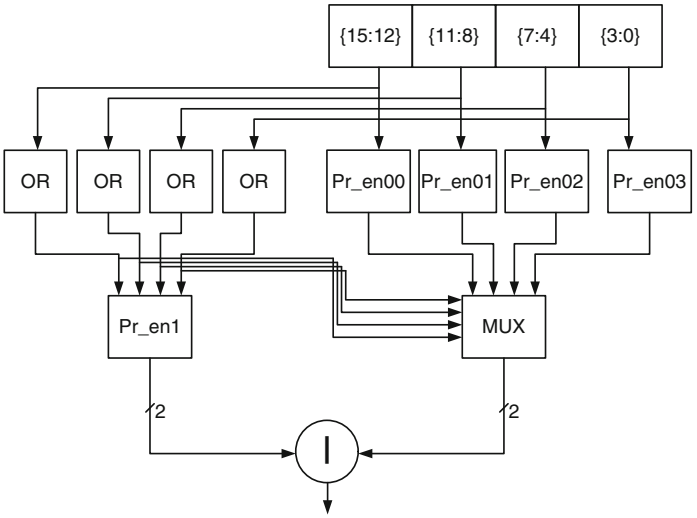

Fig. 3.3 Fast leading zero detection circuit [Alle06]

is 0, the codeword contains only bit 1s. A controller is programmed to avoid unnecessary access to a certain stage when no processing is required. Combinational logic was used, instead of LUTs. Also, like [Chang05], a 2-stage buffer at the interface is required.

This design takes up 17.2 Kgates and 5.1 Kbits of memory using 0.13 μm technology. It can operate at 250 MHz for decoding HD1080i video, and yields an average throughput of slightly larger than 1 SE/cycle due to discriminating processing. However, at the core of the CAVLD (without controller and interface), the throughput is still 1 SE/cycle.

3.2.1.3 [Lin08a]

In [Lin08a], a HW-oriented algorithm optimization and three HW optimizations have been reported. It is noted that *Level* decoding requires arithmetic operation and can be time-consuming. By analyzing the two-part *Level* decoding process and identifying the critical path, the authors moved the Suffix_Length_Detector from the second part to the first part. Therefore, the *suffixLength* required for the decoding of the next *Level* symbol can be obtained from the current symbol information and recent *level_prefix* value. The architecture of the CAVLD [Lin08a] is shown in Fig. 3.4. From the HW viewpoint, three optimizations were made: combined LUTs for *coeff_token*, *total_zeros*, and *run_before*; decoding of *Sign_T1*; and output buffer.

In S1, *coeff_token* parsing of [Lin08a], long codewords of original LUTs can be partitioned to prefix and suffix LUTs. If the parsing codeword is not in the prefix LUT, it is further looked up in the suffix LUT. Because of relatively low probabilities of long codewords and smaller prefix LUT, the average table lookup time is

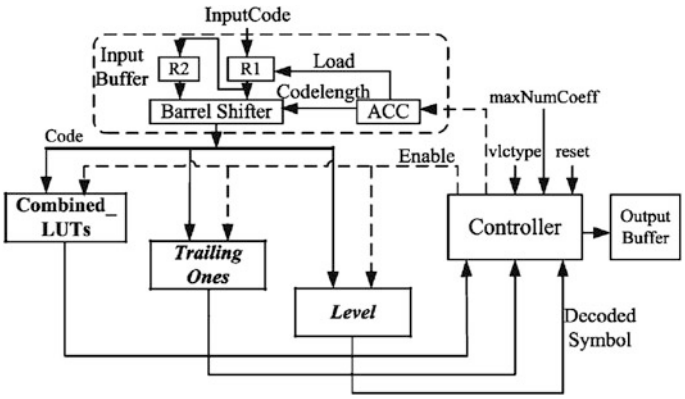

Fig. 3.4 The CAVLD architecture [Lin08a]

reduced. It was claimed that in order to disable various LUTs (5 for *coeff_token*, 15 for *total_zeros*, and 7 *for run_before*), latches have been used, and thus extra circuit area is required and unnecessary dynamic power is consumed. Since the decoding of the 3 types of SEs will not happen at the same time, it was proposed to combine LUTs for *coeff_token*, *total_zeros*, and *run_before*. The number of LUTs has been reduced to 16 and thus, the number of latches used can be reduced.

In S2, since each value "1" or "0" represents the sign "+" or "–", respectively, multiple and up to 3 sign symbols can be decoded in one cycle. Finally, the output buffer is designed to handle data while taking less area compare to the Interleaved Double Stacks (IDS) [Chang05].

Step S3 is to parse the prefix and suffix of *level* to calculate *level* values, collect and buffer all *level* values. In conventional decoding, after the parsing of *level* prefix and suffix, suffix length of current *level* is updated at the end of *level* decoding. Compared to dual-*level* decoding approach that significantly increases the critical path length, suffix length decoding of [Lin08a] is pre-calculated during prefix decoding, as shown in Fig. 3.5. Thus, *level* parsing is divided into 2 pipeline stages for the decoding of prefix and suffix.

In S4, *total_zeros* parsing of [Lin08a] focus on reducing area and power of LUT instead of enhancing throughput and coding speed, which will be discussed in later section.

S6 is to reconstruct all coefficients of the block according to *run-level* data pairs in reverse zig-zag scan order. Reconstruction of coefficient block of S6 is combined with S5 in [Lin08a]. This is because when *run_before* is known, the position of coefficient can be decided. This strategy saves processing cycles of coefficient reconstruction. Single output buffer is allocated in [Lin08a] to store the non-zero coefficient *level* values in S3. The coefficients in the buffer are moved to the correct position of coefficient block in S5. The strategies discussed in [Lin08a] can be

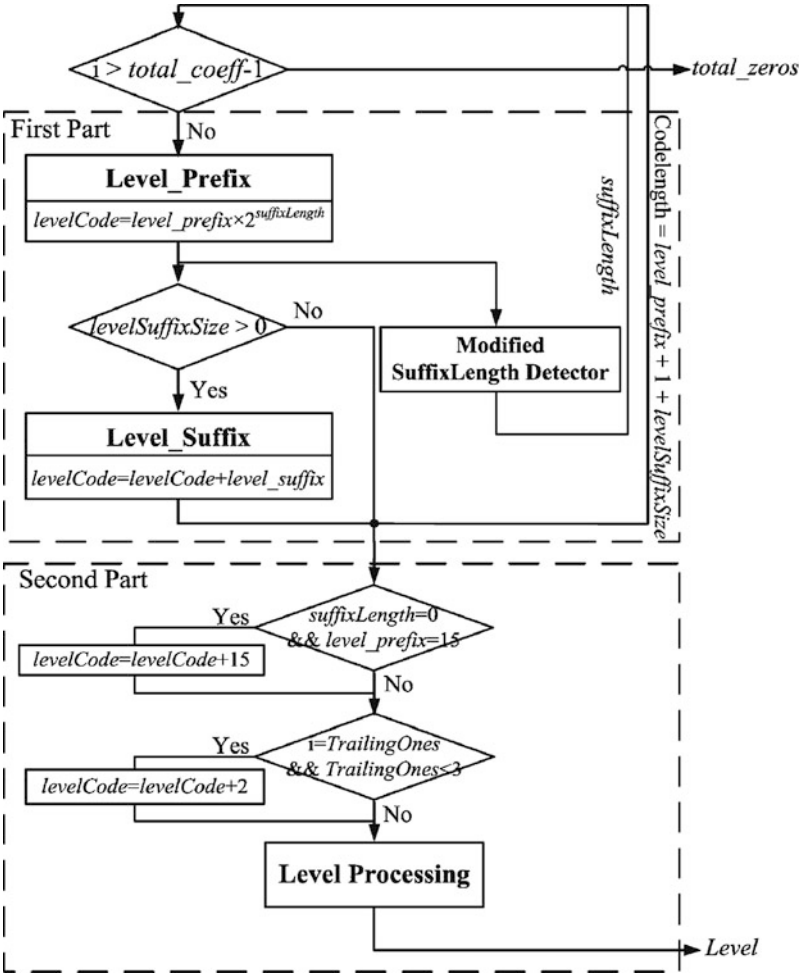

Fig. 3.5 Modified level decoding process [Lin08a]

adopted to design the interface between CAVLC decoding and the following steps of IQ and inverse integer transform.

We define, hereafter, the *processing speed* (#SE/second) as a product of throughput (#SE/cycle) and operating frequency (cycles/second). The processing speed of [Lin08a] is tripled compared to the conventional implementation, due to data dependency elimination of suffix length and pipelined decoding of *level*.

This design takes up 6.8 Kgates and 0 bit of memory using 0.18 μm technology. It can operate at 213 MHz for decoding HD1080i video, and yields an average throughput of slightly greater than 1 SE/cycle. The power consumption values are from 2 to 2.5 mW for Akyo sequence, and 4 to 5 mW for Mobile sequence.

3.2.1.4 Summary on Implementation Strategies of CAVLC Decoders

1. Parse the *coeff_token*: The original LUTs can be divided into sub-tables based on the number of leading zeros (LZ) in the codewords [Chang05, Alle06]. Sub-table selection is implemented by fast LZ detection (LZD) circuit with benefit of shorter parsing time. On the other hand, long codewords of the original LUTs can be partitioned to prefix and suffix LUTs [Lin08a]. If the parsing codeword is not in the prefix LUT, it is further looked up in the suffix LUT. Because of relatively low probabilities of long codewords and smaller prefix LUT, the average table lookup time is reduced.
2. Parse the *sign_T1s*: Parsing of *sign_T1s* is traditionally implemented in multiple cycles [Chang05, Alle06]. However, since the number of T1s has already been obtained in Step1, and each sign takes up one bit, decoding can be performed in a single cycle [Lin08a].
3. Parse the prefix and suffix of *level*: In conventional implementation, after the parsing of *level* prefix and suffix, the suffix length of current *level* is updated at the end of *level* decoding. Compared to dual-part *level* decoding approach that significantly increases critical path length, suffix length decoding of [Lin08a] is re-arranged, and pre-calculated during prefix decoding. Here, *level* parsing is divided into 2 pipeline stages for decoding of prefix and suffix.
4. Parse the *total_zeros*: This step focuses on reducing area and power of LUT instead of enhancing throughput and coding speed, which will be discussed in later section.
5. Recursively parse *run_before* values of the block: Because the codeword of *run_before* LUT is short (3 bits or less), 2 values of *run_before* are decoded in a cycle in [Alle06] with minimum addition to circuit area. When *run_before* is 0, the codeword contains only bit 1s.
6. Reconstruct all coefficients of the block: Reconstruction of the coefficient block in S6 is performed together with S5 in [Chao06, Lin08a]. This is because when *run_before* is known, the position of coefficient can be decided. The IQ operation is combined with the reconstruction of coefficient block in the 2-stage pipeline architecture in [Chao06] to save processing time. This strategy saves processing cycles of coefficient reconstruction. Single output buffer is allocated in [Lin08a] to store non-zero coefficient *level* values in S3. The coefficients in the buffer are moved to the correct position of coefficient block in S5. The strategies discussed in [Chao06, Lin08a] can be adopted to design the interface between CAVLC decoding and following steps of IQ and inverse integer transform.

3.2.2 CAVLC Encoder Design

The CAVLC encoder can be implemented in 6 steps. Only implementation strategies of the steps that improve encoding throughput and speed are discussed in this section, and several typical CAVLC encoder designs are reviewed respectively. Similar

to decoder design, maximum of 3 bits of *sign_T1s* need to be coded for each block, and it can be implemented in a single cycle [Rahman07], while strategies of other steps of encoder and decoder have some differences. The CAVLC encoder can be implemented in 6 steps as described below.

S1. Zig-zag scan the 4×4 (or 2×2 in case of chrominance) quantized transform blocks in reverse zig-zag scanned order, from the highest-indexed AC to the DC coefficient, to generate: the total number of non-zero coefficients (*TC*) and trailing ones (*T1*); sign of trailing ones (*sign_T1*); the levels (*level*) of the non-zero coefficients; total number of zeros (*total_zeros*) excluding the zeros after the last non-zero coefficient in the forward zig-zag scanned order; and runs. The 5 SEs are *coeff_token*, *sign_T1*, *level*, *total_zeros*, and *run_before*.
S2. Encode *coeff_token*.
S3. Encode *sign_T1s*.
S4. Encode *level* values of the remaining non-zero coefficients by table lookup from one of 7 LUTs.
S5. Encode total_zeros
S6. Recursively encode each *run_before* of non-zero coefficient of the block in reverse zig-zag scan order.

At the CAVLC, the inter-symbol correlations are used to remove additional statistical redundancy by allowing logically switching among LUTs of VLC depending on the previously transmitted symbols.

Entropy coding, however, involves many bit-level operations that cannot be efficiently performed by general purpose processors. Therefore, there is a need for a HW-based CAVLC accelerator.

3.2.2.1 [Kim06a]

The architecture of CAVLC design [Kim06], shown in Fig. 3.6, starts with the scanning block where the reversed zig-zag scanned coefficients received from the preceding Transform/Quantization block are extracted for *totalCoeff*, *T1*, *sign_T1*, *level*, *total_zeros*, and *run_before* syntax elements. The *coeff_token*, *total_zeros*, and *run_before* blocks collectively generate the codewords from the LUTs.

In S1, the estimated coefficient number (*nC*) is calculated to select LUT of *coeff_token*. The calculation of *nC* requires the use of the coefficient number of neighboring coded block, and the coefficient number of current coded block are to be saved subsequently. In [Kim06a], *nC* is generated in HW. The area of *nC* generation is large equivalent to 49% of the encoder design of [Kim06a]. Although additional local memory is required to store the numbers of non-zero coefficient of coded blocks, these designs benefit from the more complete functional implementations, lower memory access frequency of host processor, and lower data transfer on the system bus.

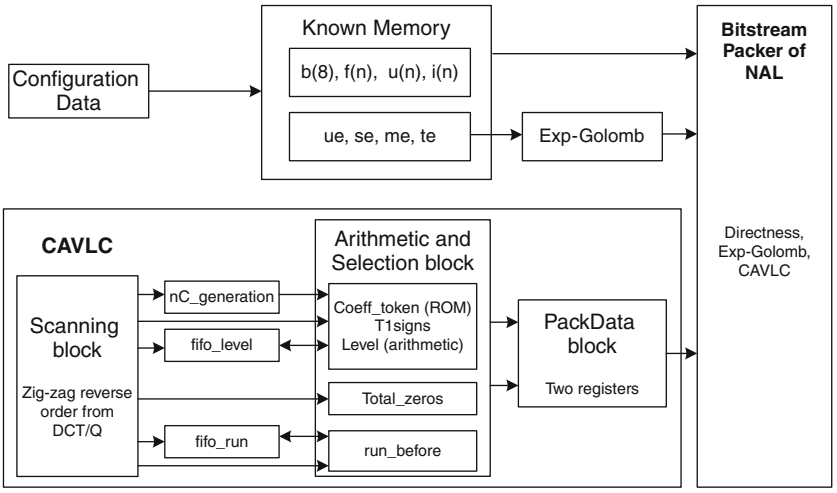

Fig. 3.6 Block diagram of CAVLC [Kim06a]

This design takes up 33.5 Kgates on the FPGA platform. It can operate at 100 MHz for encoding HD1080i video, and yields an average throughput of 16 pixels per 16 clock cycles or 1 pixel/cycle.

3.2.2.2 [Chen06c]

Identifying the burden of scanning of residues prior to coding into syntax elements, and the need to have a second pass to encapsulate data from the video coding layer (VCL) to the network abstraction layer (NAT), Chen et al. [Chen06c] proposed an CAVLC design (Fig. 3.7) having a dual-block-pipelined buffer architecture, and a coded block pattern (CBP) look-ahead scheme.

The MB information and quantized transform residues are input to the proposed CAVLC design. There are 3 levels in the entropy coding engine: symbol, codeword, and bit-stream. At the symbol level, the MB information is processed in the Exp-Golomb coding unit, while the quantized transform residues are processed in the CAVLC unit through 2 phases: scanning of residues into symbols, and coding of symbols into syntax elements. Due to the sequential nature of scanning followed by coding, the throughput is affected while the hardware resource is the same.

S1 of [Chen06c] is to zig-zag scan the 4×4 or 2×2 quantized transform blocks to generate SE values of *coeff_token*, *sign_T1s*, *level*, *total_zeros*, and *run_before*. Because the coefficient block scan is time-consuming, dual SE buffers are allocated in [Chen06c] (Fig. 3.8a) to implement a two-stage block-scan followed by SE-encoding pipeline. As shown in Fig. 3.8b, while the block-scan stage scans the coefficient blocks and generates the SE values into one buffer, the SE-encoding stage accesses the other buffer to fetch and encode SE values. In low bit-rate coding,

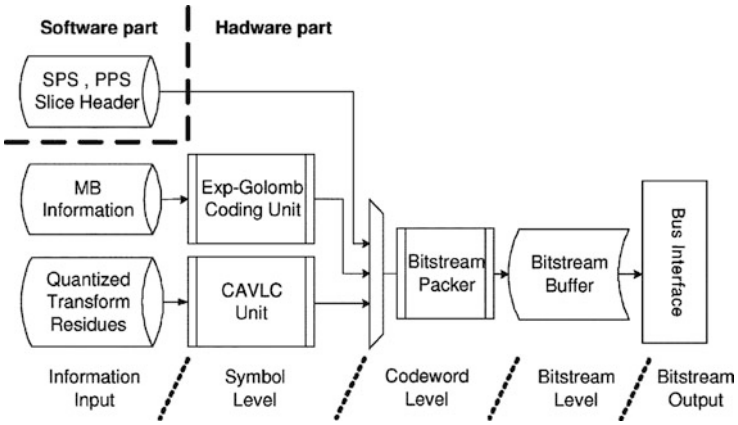

Fig. 3.7 Block diagram of H.264/AVC Baseline profile entropy coding flow

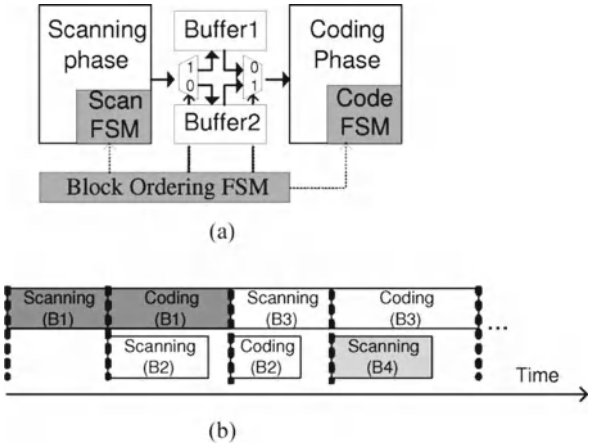

Fig. 3.8 (**a**) Dual-buffer encoder architecture and (**b**) two-stage pipeline of block scanning and coding of [Chen06c]

the ratio of non-zero coefficients is low. Thus, block-scan stage can be longer than SE-encoding stage, and pipeline efficiency can be degraded.

S4 is to encode *level* values of the remaining non-zero coefficients by table lookup from one of the 7 LUTs. Instead of storing LUTs in memory, combinational circuit is utilized to implement table lookup to accelerate level encoding in [Chen06c]. Because all level values are available in the encoder, data dependency of suffix length encoding is not critical. A dual-buffer architecture is proposed to alternatively coding and outputting the current MB in one buffer, while inputting and scanning a new MB in the other buffer. The CBP look-ahead scheme allows the by-passing of the all-zero 8×8 block to avoid scanning and coding.

This design takes up 23.6 Kgates and 2 Kbits of memory using 0.18 μm technology. It can operate at 100 MHz for decoding HDTV 720p video, and yields an average processing time of 200–500 cycles/MB.

3.2.2.3 [Tsai06]

In this design, Tsai et al. [Tsai06] approached the design from a low-power viewpoint. It was observed that memory consumption was partially due to frequent memory accesses. The Side Information Aided (SIA) Symbol Look Ahead (SLA) feature allows the CAVLC to be performed in one pass, coding only, instead of scanning and coding.

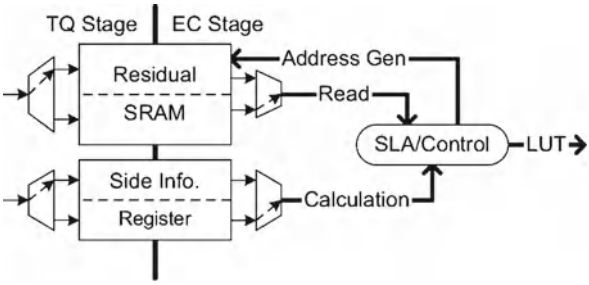

Fig. 3.9 Side information aided symbol look ahead structure of [Tsai06]

As shown in Fig. 3.9, the scan stage is removed from CAVLC encoder by providing side information of two maps of size 4×4 that indicating positions of non-zero coefficients and coefficients with absolute value 1 (abs-one). The example illustrated the SIA feature in [Tsai06] is reproduced here for demonstration. Figure 3.10 shows an example of a typical 4×4 residual block.

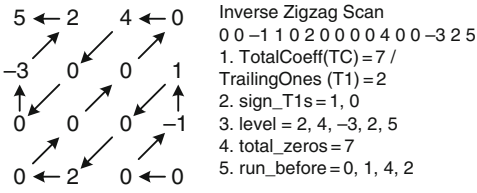

Fig. 3.10 Example of a 4×4 residual block in the CAVLC procedure

To minimize access to the residual SRAM and to provide additional information for one-pass symbol look ahead and coding, two binary maps: one for the non-zero flags, the other for the absolute-value-of-one flags are generated. Figure 3.11 shows the resulting maps of flags.

The generation of the SEs is performed mainly on these two binary maps as follows:

- TC = sum of all non-zero flags

Residuals

5	2	4	0
–3	0	0	1
0	0	0	–1
0	2	0	0

Non-Zero Flags

1	1	1	0
1	0	0	1
0	0	0	1
0	1	0	0

Abs-One Flags

0	0	0	0
0	0	0	1
0	0	0	1
0	0	0	0

Fig. 3.11 Example of the proposed SIA flags definition [Tsai06]

- *T1* = sum of all the abs_one flags located before the first one of the result of XOR(non_zero flags, abs_one flags)
- *sign_T1* = the residual SRAM with address generated by the non_zero flag's position.
- *level* = is the same as Sign_T1
- *total_zeros* = sum of all the complement of the non-zero flags, located after the first non_zero flag.
- *run_before* = difference between the previous non_zero flag's position and the current non-zero flag's position.

The operations are simple and fast. The power consumption comparison made between this design and [Chen05] shows that up to 35 and 79% reduction in power associated with logic and SRAM, respectively, have been obtained.

This design takes up 26.5 Kgates and 0 Kbits of memory using 0.18 μm technology. It can operate at 27 MHz for decoding CIF video. The power consumption was 3.7 mW.

3.2.2.4 [Chien06]

In S4 associated with *level* encoding, two processing units of *level* are allocated in [Chien06], and the suffix length generated in the 1st unit is sent to the 2nd unit for the coding of 2nd *level*.

S6 recursively encodes each *run_before* of non-zero coefficient of the block in reverse zig-zag scan order, a dual-SE coding of *run_before* is supported in [Chien06] by arithmetic coding instead of LUT. The block coding cycles are significantly reduced with such multi-SE encoding strategy.

3.2.2.5 [Rahman07]

Rahman et al. [Rahman07] proposed a CAVLC block for handling mobile video application in real-time. This design attempted to reduce circuit area by using split VLC LUT, and *Arithmetic Table Elimination* (ATE) technique (Fig. 3.12).

The ATE block is used to generate the level codes instead of using LUTs. A counter and an accumulator compute the *run_before* and ZerosLeft for LUT matching (for run_before codes). The *run_before* is the output of the counter which is

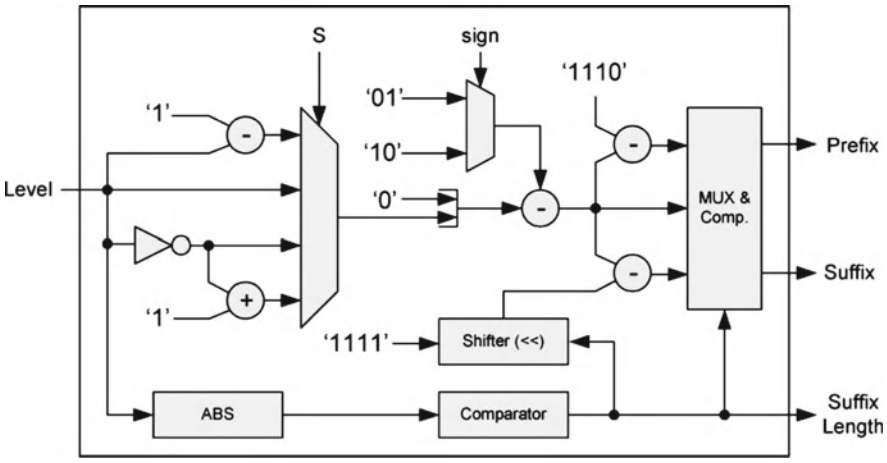

Fig. 3.12 Architecture of the ATE for level code generation [Rahman07]

subtracted from the current value of the accumulator to find the ZerosLeft for the next nonzero coefficient. The accumulator is initialized before with the value of *total_zeros* of the current input block.

The design was implemented on the Virtex II FPGA platform using 6.8 Kgates running at 50 MHz for decoding CIF/QCIF at 30 fps.

3.2.2.6 Summary of Implementation Strategies of CAVLC Encoder Designs

The functions of a CAVLC encoder can be implemented in 6 steps. Only implementation strategies of the steps that improve encoding throughput and speed are discussed in this section.

1. Zig-zag scan the 4×4 (or 2×2) quantized transform blocks to generate SE values of *coeff_token*, *sign_T1s*, *level*, *total_zeros*, and *run_before*. Because the coefficient block scan is time-consuming, dual SE buffers are allocated in [Chen06c] to implement a two-stage block-scan followed by SE-encoding pipeline. While the block-scan stage scans the coefficient blocks and generates the SE values into one buffer, the SE-encoding stage accesses the other buffer to fetch and encode SE values. In low bit-rate coding, the ratio of non-zero coefficients is low. Thus, block-scan stage can be longer than SE-encoding stage, and pipeline efficiency can be degraded.
2. Another technique using *side information aided* (SIA) *symbol look ahead* (SLA) is proposed in [Tsai06] to remove the scan stage from CAVLC encoder by providing two additional 4 × 4-b maps: one indicates the positions of non-zero coefficients, and the other indicates the coefficients with absolute value 1 (abs-one). As S1 is removed from [Tsai06], average throughput of 1.7 SE/cycle is achieved.

3. The estimated coefficient number (*nC*) is calculated to select LUT of *coeff_token*. The calculation of *nC* needs to utilize the coefficient number of neighboring coded block, and coefficient number of current coded block needs to be saved. In [Chen06c, Kim06a], *nC* is generated in HW. The area of *nC* generation occupies 16.3 Kgates, equivalent to 49% of the encoder design of [Kim06a]. Although additional local memory is required to store the numbers of non-zero coefficient of coded blocks, these designs benefit from more complete functional implementations, lower memory access frequency of host processor, and lower data transfer on the system bus.
4. Encode *sign_T1s*. As maximum of 3 bits of *sign_T1s* need to be coded for each block, it is implemented in single cycle in [Rahman07].
5. Encode *level* values of the remaining non-zero coefficients by table lookup from one of 7 LUTs. Instead of storing LUTs in memory, combinational circuit is utilized to implement table lookup to accelerate *level* encoding in [Chen06c, Hu08], etc. Because all *level* values are available in the encoder, data dependency of suffix length encoding is not critical. Two processing units of *level* are allocated in [Chien06], and the suffix length generated in the 1st unit is sent to the 2nd unit for the coding of 2nd *level*.
6. Encode *total_zeros*: no particular technique implementated.
7. Recursively encode each *run_before* of non-zero coefficient of the block in reverse zig-zag scan order. Dual-SE coding of *run_before* is supported in [Chien06] by arithmetic coding instead of LUT. The block coding cycles are significantly reduced with such multi-SE encoding strategy.

3.2.3 Comparisons of Synthesized CAVLC Codec Designs

Implementation strategies that reduce circuit area and reduce power consumption of CAVLC codecs are investigated in this section, and the synthesized results of different designs are compared.

In CAVLC decoder design, latches are allocated in [Lin08a] to control the table lookup operation of both prefix and suffix LUTs during codeword parsing. Sub-tables of *total_zeros* are combined with similar sub-tables of *coeff_token* [Lin08a]. As S1 and S4 do not occur simultaneously, the number of LUTs can be reduced. Clock gating is also applied to reduce power of unaccessed sub-tables. Forty percent reduction of power consumption is achieved in [Lin08a], and it only consumes 2.5 mW at 100 MHz operating frequency using 0.18 μm CMOS process.

In encoder design, to achieve complete statistical (entropy) coding functions of the Baseline profile of H.264/AVC, Exp-Golomb coding circuits are integrated with the CAVLC encoder in [Kim06a]. Compared to the designs that only implement CAVLC functions, no further bit stream packing circuit is needed in [Kim06a], and the coded bit stream can be output directly to the NAL layer. In [Tsai06] with SLA encoding, scan procedure is removed compared to [Chen06c] and the area of the symbol buffer of the dual-block pipeline architecture is saved in [Tsai06], while the encoding throughput is unchanged. Power consumption of SRAM in [Tsai06]

reduces to 21% of [Chen06c] because of less memory allocated and lower frequency of SRAM read access of residual block. Limitation of SLA is that: larger circuit area is required, compared to other designs. Additionally, the side information maps still need to be calculated by the functional block of transform and quantization step, and require additional circuits. A reasonable scheme is to utilize parallel comparators to generate the side information of SLA after transform and quantization. For *coeff_tokens* encoding of S2, VLC tables are split and SEs are mapped to prefix and suffix codewords before final concatenation in [Chen06c], and 5% area of LUT is reduced. In [Rahman07], the arithmetic table elimination (ATE) reduces the circuit area of *level* encoding by using simple arithmetic coding circuit instead of fixed-length LUTs, compared to the other designs.

Table 3.1 lists the synthesized CAVLC decoders and encoders with their coding performance or functions. Among the CAVLC decoders, significantly larger circuit area is required in [Alle06] compared to other designs to achiever higher operating frequency of 250 MHz. Design in [Lin08a] achieves fast decoding speed with considerations of process technology (0.18 μm), throughput (#SE/cycle) and operating frequency, and its circuit area is also minimized compared to the other reported designs. Compared to [Chang05], design in [Lin08a] enhances coding speed and reduces circuit area. It is noted that a design with 2X throughput and critical path length in the range of (Y, 2Y), is not significantly faster in coding speed (#SE/second) compared to a design with 1X throughput and 1Y critical path length, everything else being the same.

Among the reported CAVLC encoders, the most functionally complete CAVLC encoder is implemented in [Chen06c]. Using SLA, memory access frequency and the related power consumption of [Tsai06] is significantly lower than [Chen06c], while its throughput of 1.7 SE/cycle is significantly higher than [Chen06c] as the coefficient scan and SE generation is removed from the encoder. FPGA design in [Rahman07] is also reported using only 6.9 Kgates. The maximum coding speeds of the encoders are not comparable with the provided synthesized circuits.

Table 3.1 Comparison of synthesized CAVLC decoder and encoder designs

Design	Process Tech	Throughput (SE/cyc)	Max. clock (MHz)	Coding speed (M SE/s)	Circuit area logic | mem Kgates (Kbits)
			Decoder		
[Chang05]	0.18 μm	1	177	177	9.9 | 1.2
[Alle06]	0.13 μm	> 1	250	> 250	18.7 | 5.1
[Lin08a]	0.18 μm	> 1	213	> 213	6.8 | no mem
			Encoder		
[Chen06c]	0.18 μm	1	100	100	23.6 | 11.4
[Tsai06]	0.18 μm	1.7	27 (not max)	> 46	26.6 | not reported
[Rahman07]	FPGA	0.33	50	16.5	6.9 | not reported

3.3 CABAC Decoder and Encoder IP Designs

CABAC has its name from two major concepts: arithmetic coding and time-varying statistics adaptability. The arithmetic coding involves the recursive assignment of symbols into durations which correspond to the symbols' probabilities of occurrence.

From algorithmic viewpoint, in order to provide a complete IP of CABAC decoder, the 3 decoding steps need to be implemented, including BAD, BM, and CM. For CABAC encoder design, all 3 functional steps need to be implemented, including BN, CM, and BAC. Partial implementation of these functional steps will reduce the integratability of an IP block in the top-level H.264/AVC codec, and degrade the coding performance of IP. General approaches of high performance HW architectures include pipelined de/coding, data pre-fetch to reduce access latency, cascaded multiple-symbol processing units with higher throughput, etc. Detailed implementation strategies of CABAC decoder and encoder will be discussed separately in the following two sections.

3.3.1 CABAC Decoder Designs

Block diagram of CABAC decoder of H.264/AVC is illustrated in Fig. 3.13, including the following 3 functional steps: (1) binary arithmetic decoding (BAD), (2) context model selection and access (CM), and (3) binarization matching (BM).

As shown in Fig. 3.13, coded bit-stream from H.264/AVC encoder is input to BAD, in which regular bin (RB) and bypass bin (BB) are decoded in the regular bin decoding engine and bypass bin decoding engine, respectively. For RB decoding, one context model is selected for each bin in the CM based on the decoding SE type, bin index (*binIdx*) and decoded bin values provided by BM and SE values of decoded SEs of the neighboring decoded MBs. Each context model can be accessed according to *CtxIdx*, which is the sum of context offset (*CtxOffset*) and

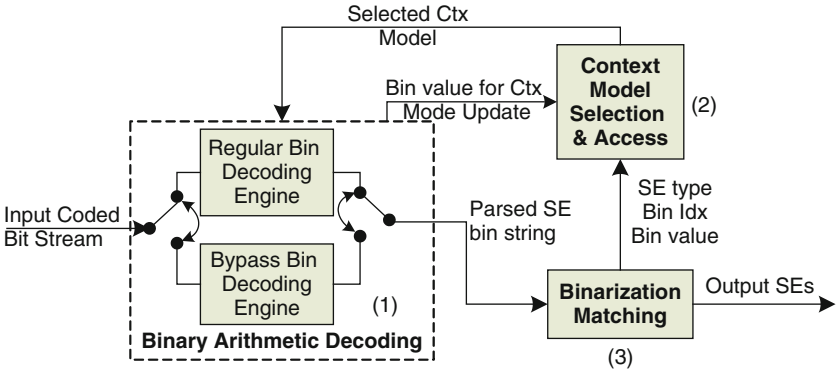

Fig. 3.13 Block diagram of CABAC decoder

context index increment (*CtxIdxInc*). *CtxOffset* locates one context model within the set of context models of one SE type. *CtxIdxInc* depends on values of coded bin or coded SE in the neighboring coded MB. The decoding *Range* and *Offset* are updated based on context model, and the bin value is decoded in BAD. Based on the decoded value of regular bin, the corresponding context model is updated to adaptively adjust the probability estimation of RB. Decoded bins are parsed in BM, where the decoded bin string is compared with the bin string patterns of the same SE type to decide whether the decoding of current SE completes. Decoded SEs are sent to the following decoding steps of H.264/AVC to reconstruct the video sequence. Data dependency exists in a long loop to select a proper context model for next RB (BAD -> BM -> CM -> BAD). Design challenge of CABAC decoder is to undo the loop, and enhance decoding throughput.

3.3.1.1 Binary Arithmetic Decoding (BAD)

In BAD, input bins of RB, BB, and TB (terminate bin) are decoded separately, and context model is provided by CM for RB decoding. During the decoding of some types of SE, the current decoded bin is used to select the context model of next decoding bin. Because of data dependency aforementioned, it is difficult to achieve multi-bin decoding per cycle in such situation. However, when the context access pattern is fixed, such as decoding of residual SEs including *significant coefficient flag* (SCF), *last SCF* (LSCF), and coefficient level, cascaded decoding units for multiple bins achieve a throughput of 2 bin/cycle (2 RB, 2 BB, or 1 RB and 1 BB) [Yu05, Chen07a, Li07a]. The cascaded bin double-bin decoding architecture of [Yu05] is shown in Fig. 3.14.

To reduce the critical path of 2 RB decoding, two possible R_{LPS}_4 (4 possible values of *Range* of LPS) values of RB2 are pre-calculated during RB1 decoding in [Yu05], and the correct R_{LPS}_4 value is selected for RB2 when RB1 is decoded. Dual-RB decoding of [Kim06b] constantly predicts that RB1 is MPS, and begins RB2 decoding, and the critical path of RB decoding is shorter than that of [Yu05]. However, throughput of [Kim06b] is 0.56 bin/cycle, because if the prediction is incorrect, the decoded RB2 is discarded. The limitation of dual-bin decoding scheme is that although throughput of residual SE decoding can be increased to 2 bins/cycle in some situations, critical path length also increases by a ratio in the range of (1Y, 2Y]. For the SEs that can only be decoded at a throughput of 1 bin/cycle – including non-residual SEs and coded block flag (CBF), decoding time is prolonged. The overall performance improvement is not significant, especially in low bit-rate coding, with consideration of significantly larger area to support cascaded decoding engines and more complex control logic.

3.3.1.2 Context Model Selection and Access

The context model of the decoding RB is selected in this step and accessed from context RAM according to the calculated *CtxIdx*. In order to reduce the context RAM access delay, a group of context models can be pre-fetched into the local

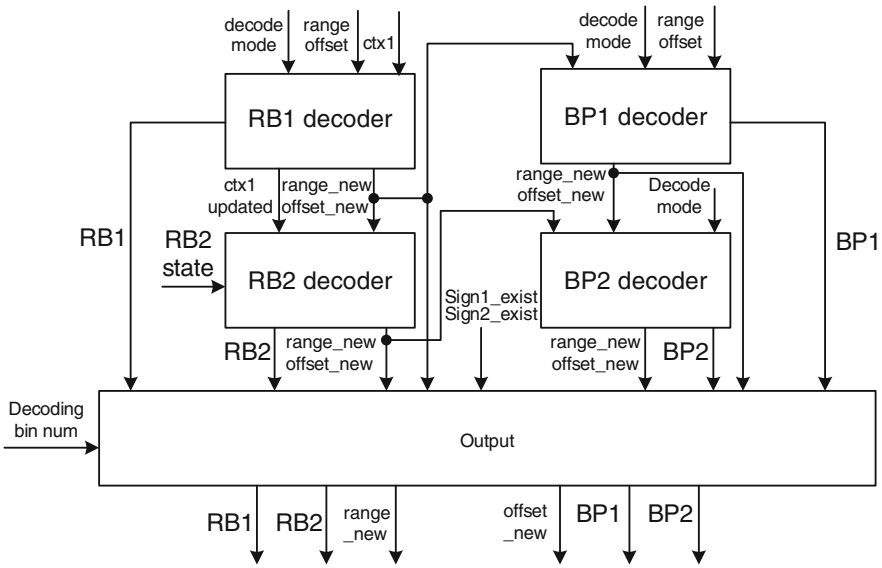

Fig. 3.14 Double-bin decoding architecture of [Yu05]

buffers [Yu05, Yi07]. Because the context model selection and access of coded block pattern (CBF) requires multiple cycles [Chen05], *CtxIdx* of the next CBF is pre-calculated during the decoding of current 4×4 block [Chen07a]. The decoding bin percentage of the SEs of significant map including SCF and LSCF is also significant. To accelerate significant map decoding in the dual-bin decoding architecture, context models of LSCF are stored in a separate SRAM and possibly to simultaneously access context models of both SCF and LSCF SEs [Chen07a]. As shown in Fig. 3.15, a separate context RAM is used to LSCF context models.

The techniques of context model pre-fetch, *CtxIdx* pre-calculation, and parallel access of multiple context models are beneficial to reduce context access delay.

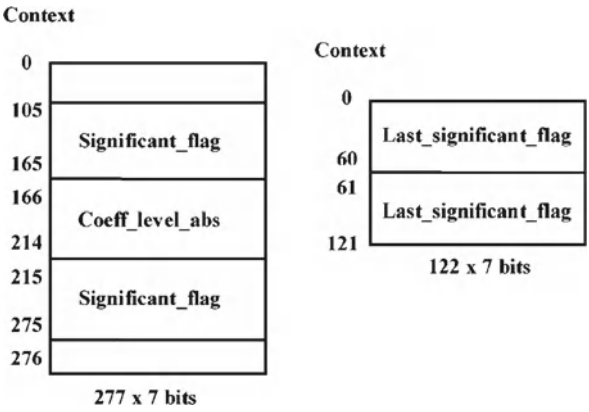

Fig. 3.15 Map of context models of context RAM [Chen07a]

3.3.1.3 Binarization Matching and SE Generation

BM is the inverse binarization of CABAC encoding. It can be controlled by a FSM to generate the type of next decoding SE. The corresponding LUT of the SE is accessed to match the decoded codeword. The parsed SE is output in this step. Because BM is one decoding stage of CABAC decoding pipeline, several strategies are discussed in Sect 3.3.1.4) to reduce pipeline stall and enhance decoding throughput, including parallel processing of BAD and BM.

3.3.1.4 Solutions to Pipeline Hazards of CABAC Decoding

A FPGA-based acceleration of critical decoding loop of context model access (CA) of CM and BAD is proposed in [Eeckhaut06]. The acceleration is achieved by mapping sequential operations of memory access and table lookup of CA and BAD into pipeline stages with data pre-fetching and data forwarding. A throughput of 1 bin/cycle is claimed. However, the data dependency between the decoded RB and the context model of the next decoding RB is not totally removed. Actual throughput of the decoder is influenced by both the maximum processing speed of CA and BAD and the architecture of BM and context model selection.

To enhance decoding throughput, BAD and BM are designed in the same pipeline stage in [Yi07], which is also adopted in [Son08] and [Shi08]. The benefit of combining BAD and BM in the same stage is that the SE type and *CtxIdx* of the next decoding bin can be decided by BM after the bin is decoded by the BAD. Therefore, one cycle is saved in RB decoding. A small context buffer named CMR is also allocated in [Yi07] that can buffer a maximum of 8 context models of one SE type. With the assists of CMR, context model selection (*CtxIdxInc* calculation) and operation of context models loading from SRAM (CL) based on *CtxOffset* can be designed as two parallel units in the same pipeline stage. Another benefit of CMR as discussed in [Yi07] is that SRAM write operation of updated context model (CU) can be separated from the pipeline stages and the conflict of CL and CU on the same SRAM position is solved by data forwarding. The limitation of [Yi07] is that two cycles of pipeline stall cannot be avoided for CU and CL operations when the type of decoding SE changes. For large ratio of decoding SEs, number of bins per SE of is small. Thus, pipeline stall occurs frequently, and it takes an average of 3.9 cycles to decode one bin. In [Son08], the pipeline stall frequency is reduced by predicting the type of next SE. Prediction is made according to the SE values of neighboring coded blocks. With the SE prediction technique applied to the 2-stage decoding pipeline in [Son08], over 60% decoding cycles are reduced compared to [Yi07]. The limitation of SE type prediction is that if the prediction is incorrect, one cycle of stall will still be introduced for the loading of correct context models.

To eliminate pipeline stall of CABAC decoding, 4-stage decoding pipeline is reported in [Shi08] that achieves a decoding throughput of 1 bin/cycle for RB decoding as shown in Fig. 3.16.

Context model selection and CL are separated to stage 1 and stage 2 of the pipeline. During the decoding of current SE, the first context model of next SE

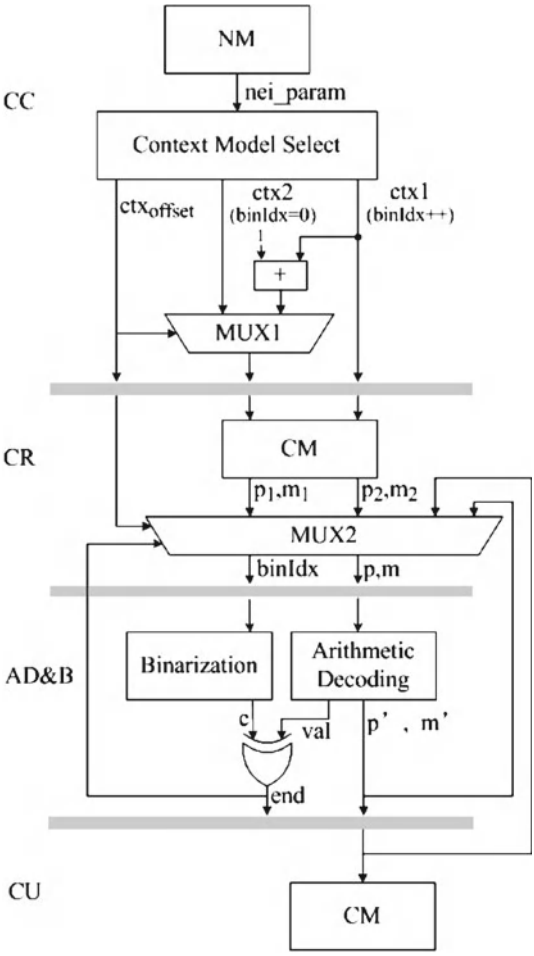

Fig. 3.16 Four-stage CABAC decoding pipeline of [Shi08]

is selected in stage 1 and loaded from context RAM in stage 2. In stage 2, two context models are read separately from two the context RAMs in parallel in one cycle. The context model of current SE is read from one RAM containing entire context models, while the other context model of the next SE is read from a small RAM that only stores the first context model of each SE. The decoded bin in stage 3 of BAD and BM decides whether a new SE will be decoded in the next cycle. Based on the decision, one of the two loaded context models is selected in stage 2 for the next decoding RB of stage 3. Updated context model of decoded RB is written back to context RAM in stage 4. Compared to the other designs, by pre-fetching context model of next SE during current SE decoding, pipeline stall is avoided in [Shi08] when decoding SE changes. Speedup 6.6 times is reported in this 4-stage pipeline, compared to conventional design. Throughput is enhanced at the cost of doubling of context RAM access frequency, and complicated multi-branch *CtxIdx* calculation.

3.3.2 CABAC Encoder Design

The block diagram of CABAC encoder of H.264/AVC is shown in Fig. 2.4. CABAC encoder consists of 3 functional steps: (1) *Binarization* to map SE value to bin string; (2) *Context model selection and access* to select the proper context model (probability model) for regular bin (RB) according to context index (*CtxIdx*), and update context model after RB coding; (3) *Binary arithmetic coding* (BAC) to encode each bin by dividing coding interval of [*Low*, *Low* + *Range*) and selecting one of the two subintervals $Range_{LPS}$ and $Range_{MPS}$ based on whether the bin is MPS or LPS, and probability of MPS. For the bin with equal probability, bypass bin (BB) coding route is used. The upper bins of *Low* are shifted out as coding result when the bin values are fixed.

Compared to CABAC decoding, CABAC encoding has lower level of data dependency. It is because binarization is an independent functional step with no feedback loop from CM or BAC to binarization compared to the BM in CABAC decoding. Inaddition, context model selection of CM does not depend on the result of context access (CA) of CM and BAC. After binarization, *CtxIdx* calculation of each RB of bin string of SE can be carried out, which is used to select context model. Therefore, binarization and context model selection can be pre-calculated and separated from CABAC encoding pipeline. Because only CA and BAC are the two necessary coding steps for each RB, it is easier to design encoder pipeline with less frequent pipeline stalls compared to the previously discussed decoder designs. CABAC encoder designs of H.264/AVC are reviewed as follows.

Reported by Li et al. [Li06a], a dynamic pipeline scheme is used to reduce pipeline bubbles. However, because coding interval subdivision and renormalization are separated in two pipeline stages, data dependency of the two stages causes frequent pipeline stall and a throughput of 0.59 bin/cycle is obtained.

A CABAC encoding core of context access and BAC is proposed in [Kuo06] with low power design consideration. A variable bit-length tag cache and a register file of 72×7 bits are allocated in the design to reduce SRAM access frequency and the associated power consumption. Technique such as clock gating is also adopted to reduce power consumption of the tag cache. The throughput is less than 1 bin/cycle because of additional cycles for RAM access of context models when cache miss occurs.

An arithmetic encoder that supports both JPEG2000 and H.264/AVC is proposed in [Flordal06]. To encode 2 RBs per cycle, the *inverse multiple branch selection* (IMBS) is adopted, and concurrent *Range* update is achieved by pre-calculating all of the possible output values of first *Range* before the correct value is selected. IMBS results in large circuit area for the pre-calculation of all possible branches. Because data forwarding mechanism in context model access is not built, the pipeline will stall when two successive RBs access the same context model. Because coding of bin with equal probability (BB) is not implemented, the design cannot fully support BAC. Furthermore, only estimated value of average throughput of the encoder is reported in [Flordal06].

In [Kuo06, Li06a], binarization and context model selection are left to be run on the host processor. This requires additional processing power and memory bandwidth at the host system.

Chen et al. [Chen07b] combines the *CtxIdx* calculation and binarization into one module of a 3-stage pipeline. Similar to [Flordal06], pipeline stalls when the same context model is successively accessed during RB encoding. As a result, the achievable throughput is 0.56 bin/cycle. Context selection is not completely implemented in the design, and binarization is overlapped in the encoder and host.

Compared with the previous CABAC encoder designs, Liu et al. [Liu07a] implements context model selection (*CtxIdx* generation) fully in HW with reference to the stored SEs of neighboring coded MBs. Although a 3-stage pipelined module is designed to accelerate the RAM access of coded SEs from neighboring blocks, the encoding pipeline is still stalled for 4 cycles in each access. The resulting throughput of the encoder is 0.67 bin/cycle.

A full hardware CABAC encoder is proposed by Lo et al. [Lo07] with an average throughput of more than 1 bin/cycle. Throughput is enhanced because cascaded arithmetic coding engines are allocated in BAC to support coding of 2 bins/cycle of residual SEs including significant map and *level* values of coefficients. To support coding of 2 bins/cycle of significant map, context models of SCF and LSCF are allocated in a separate SRAM, which enables reading of 4 context models (2 pairs of SCF and LSCF) per cycle. Although it is claimed that the coding speed can be doubled, the speed up ratio is actually below 2. This is because the critical path length of BAC is extended, encoding time of non-residual SEs and CBF is extended because throughput of this part of SEs is still 1 bin/cycle and clock frequency is lower. In addition, because only one context model of *level* can be accessed per cycle, the throughput of level coding will be only 1 bin/cycle in some situations. Circuit area is also increased to support higher coding throughput.

The multi-bin arithmetic encoding is investigated in [Chen06d]. To enable multiple context model access in each cycle, SRAM banks are utilized to provide sufficient SRAM ports for parallel access of context models from different banks. Data forwarding architecture is utilized to avoid read and write conflicts of SRAM access and pipeline bubble when the same context model is accessed in successive cycles. The limitation of this scheme is that a highly complicated context selection block is needed to prepare multiple context models for the context access pipeline in each cycle. In addition, cascaded units coding interval subdivision and selection are needed and the critical path length is significantly longer compared to single bin coding scheme. The throughput is degraded because of data dependency in the binarization and context model selection.

In all CABAC encoder designs discussed above, *Rate Distortion Optimization* (RDO) is not supported. It is analyzed in [Nunez06] that the high computational requirement of CABAC is largely due to the support of RDO in the H.264/AVC encoding system. CABAC encoder design support for RDO is considered in [Nunez06], with supports of context state backup and restoration operations when RDO mode changes.

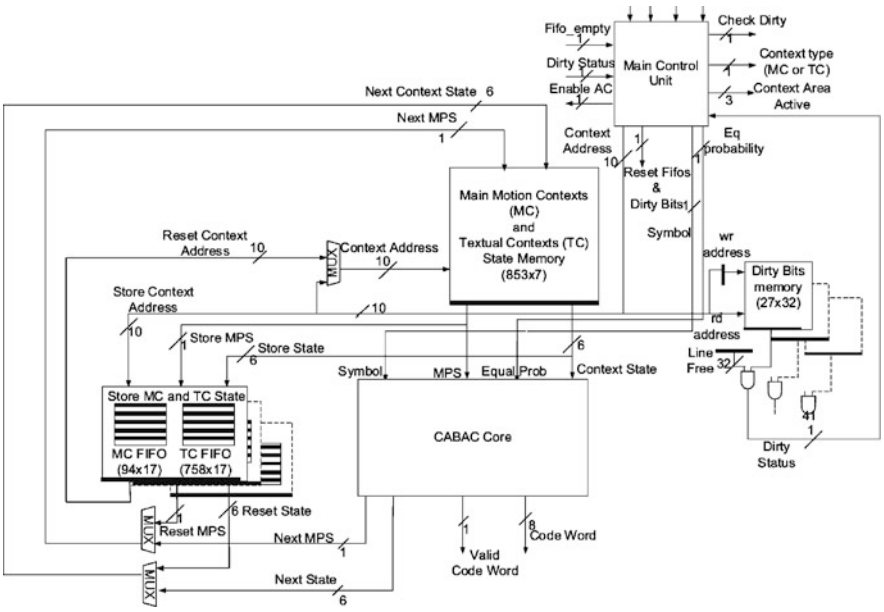

Fig. 3.17 Rate-distortion optimization assisted CABAC encoding with circuits of context state backup and restoration [Nunez06]

As shown in Fig. 3.17, three large FIFO buffers are allocated in the design to back up the intermediate states of context models during RDO mode decision procedure. However, because the throughput of FIFO read/write is one context model per cycle, timing delay is long for the operations of context state backup and restoration. In addition, large FIFO buffers occupy large circuit area. Because one context model is accessed from context RAM in the design, and there is no mechanism to reduce RAM access by allocating local cache or context buffer, the access frequency and power consumption of the context RAM are significant. In [Nunez06], the encoding core only focuses on acceleration of context access and BAC. Binarization and context model selection are not implemented. RDO is not fully supported in [Nunez06] because context model selection related RDO operations are not implemented including the state of coded SEs backup and restoration of current coding MB during RDO mode decision procedure.

Osorio et al. proposes the first arithmetic coding architecture of H.264/AVC of throughput of 1 bin/cycle, utilizing multi-stage pipeline to accelerate CABAC coding steps including context access, BAC, and bit packing [Osorio04]. The coding throughput is further enhanced in their latest design [Osorio06], in which *Range* and *Low* are updated in separate arithmetic coding pipeline stages with duplicated computational resources to significantly increase throughput to around 2 bins/cycle. Figure 3.18 shows the CABAC implemented by [Osorio06].

A small cache of 16 context models is utilized to buffer context models read from RAM. The context access delay is reduced, because for RBs of residual SE,

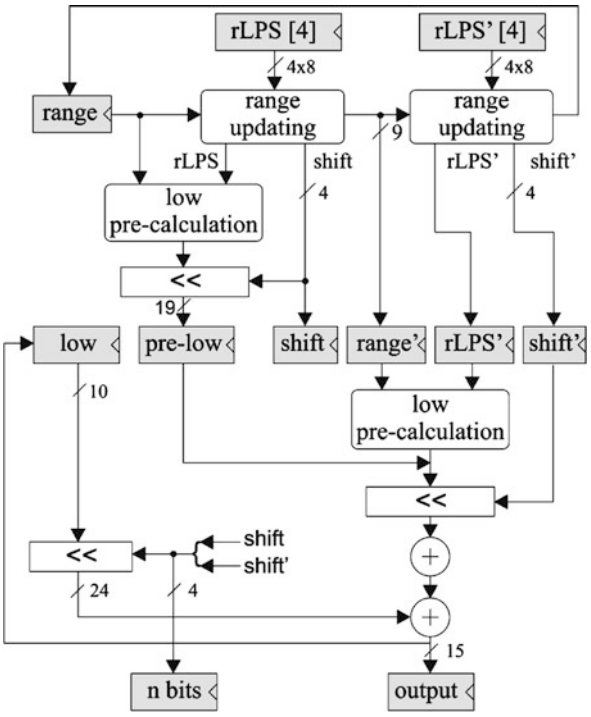

Fig. 3.18 Arithmetic coding interval update with double bins per cycle throughput [Osorio06]

context models are pre-fetched to the cache before bin coding. In each cycle, a pair of two RBs with the corresponding context models is processed by BAC. For the non-residual SEs and CBF, binarization, bin pair preparation, and context model selection are assumed to be performed by the host processor. It is noted that these operations take up a large percentage of total CABAC encoding instructions. To enable data pairing operation, additional computational cost is further assumed by the host processor. To support RDO, two RAM blocks are allocated to store original and updated context models during RDO coding. However, P8×8 RDO coding mode is not supported in [Osorio06], which is critical to the efficiency of inter frame coding of H.264/AVC. In Chap. 7, limitations of [Osorio06] will be analyzed in details.

3.3.3 Comparisons of Synthesized CABAC Codec Designs

Table 3.2 lists the synthesized CABAC decoders and encoders with impressive coding performance or functions. Other designs [Lo07, Chen06d] are not reported because of unavailable data at writing time or not good enough of a design.

For CABAC decoder design, synthesis results of 3 designs of 0.18 μm process are listed. Although 1–3 bins can be decoded by BAD block in [Yu05], the design only focuses on acceleration of BAD. With BM and CS are handled by the

Table 3.2 Comparison of synthesized CABAC decoder and encoder designs

Design	Process tech	Throughput (bin/cycle)	Max. clock (MHz)	Coding speed (Mbin/s)	Circuit area (gates) of logic functions
			Decoder		
[Yu05]	0.18 μm	1~3 (only of BAD)	149	Average N/A	0.3 mm^2 (partial)
[Shi08]	0.18 μm	1.27	200	254	29.0 K
			Encoder		
[Li06a]	0.35 μm	0.59	150	89	4.57 K (CA and BAC)
[Osorio06]	0.35 μm	1.9 ~ 2.3	186	372	19.4 K gates
[Chen07b]	0.15 μm	0.56	333	186	13.3 K(no calc of *CtxIdxInc*)
[Liu07a]	0.13 μm	0.67	200	134	34.3 K gates
[Tian09]a	0.35 μm	1	186	186	31.2 K (full HW w/RDO)
[Tian09]b	0.13 μm	1	578	578	44.6 K (full HW w/RDO)

host processor, the average throughput of CABAC decoder [Yu05] will be significantly lower than the reported BAD throughput. Complete CABAC decoding is implemented in [Shi08]. Data dependency among the decoding steps is efficiently reduced in the pipeline stages in [08], and high throughput of 1.27 bpc and 200 MHz operating frequency are achieved.

For CABAC encoder design, throughput of 1 bpc and 578 MHz operating frequency of [Tian09]b are faster than that of [Chen07b, Liu07a] of similar process technology (over 1.5 times higher in throughput). This is because unit CA and BAC of [Tian09] are properly partitioned into 3 pipeline stages with data pre-fetch and pre-calculation techniques. The data forwarding path of context RAM access is enabled in CA to avoid RAM read and write conflicts in [Chen07b] when the same context model is successively accessed. Although the context model selection is also fully implemented in [Liu07a], throughput of [Liu07a] is low because of frequent pipeline stall when accessing the coded SEs of neighboring MBs.

For the same 0.35 μm process, the coding speed of [Tian09]a at 186 Mbin/s is significantly higher than 89 Mbin/s of [Li06a], as the context model is pre-fetched, and the risk of pipeline bubble is removed. In addition, binarization and context model selection are left to be run on the host processor in [Li06a]. This implies additional computation and memory bandwidth must be supported by the host system. Higher coding throughput (1.9~2.3 bpc, and 1.5 times speed up of bin/s) is achieved in [Osorio06], as two RBs can be processed sequentially in BAC in the same cycle. The limitation of design reported in [Osorio06] is that it focuses on HW acceleration of residual SEs. Binarization, context model selection, and data pairing of non-residual SEs are generated by host processor. The computational complexity

of [Osorio06] on the host processor is significantly higher than [Tian09]. Pipeline stall can occur and throughput can be dropped when the host processor cannot generate data in the same pace as it is processed by the encoder. For the design that supports encoding of two RBs/cycle of residual SEs including significant map and level, because only one context model of level can be accessed in each cycle, the throughput of level coding will only be 1 bpc in some situations, and the average throughput of residual SEs is less than 2 bpc.

The circuit area of [Tian09] is larger compared to other reported designs, because more complete CABAC functions are implemented including binarization, context model selection, and RDO. Large circuit area is allocated for the RDO operations and *CtxIdxInc* calculation which are not implemented in most reported designs.

3.4 Summary of Implementation Strategies CABAC Encoder and Decoder

Implementation strategies of CABAC encoder and decoder designs have been investigated. For CABAC decoder designs: To accelerate decoding procedure, strategies of cascaded processing units, data pre-calculation, and reducing pipeline stage number are useful. To reduce context model access delay, strategies of context model prefetch and local buffering, separating context RAM tables for multi-context models accessing, and *CtxIdxInc* pre-calculation are efficient. In CABAC encoder design, it is beneficial to allocate multiple pipeline stages to increase encoding speed. Multi-bin encoding is a choice to enhance throughput. However, performance improvement of it is not significant because of lower clock frequency, non-constant throughput, and larger area of the control logic for multi-bin encoding.

Generally speaking, the most important strategies of entropy codec designs include: (a) pipeline acceleration by data dependency removing (6.6 times speedup [Shi08]), (b) table lookup optimization (40% power reduction [Lin08a], and 65–88% memory access reduction [Moon05]), and (c) critical path reduction by data pre-fetch and pre-calculation. Functional completeness and processing time reduction are the two high-priority targets of many designs, while reducing power consumption is another consideration of some designs.

For CABAC encoder designs, the implementations discussed above, excluding [Tian09], have several limitations in general, including:

1. HW encoder function is not complete, which costs high computation complexity on the host processor and limits the performance of CABAC design;
2. Data dependency of coding steps is not effectively removed, so the encoder cannot be implemented in a full pipeline structure, and the throughput is low;
3. RDO is not supported in these designs or at least not efficiently supported, which costs high computation burden on the host processor, and requires large bandwidth to support backup and restoration of CABAC coding states;

4. Memory access frequency and related power consumption are high in most designs because of single context model access from context RAM, and encoder power reduction techniques are not reported in most designs;
5. Reusability and integratability of encoder IP are not considered.

In the following chapters, design of a typical entropy coder [Tian09] with comprehensive coding functions and stable high performance is proposed to solve the limitations aforementioned and achieve research objectives listed in Chap. 1. Implementation strategies of the proposed CABAC encoder will also be discussed, and comprehensive performance comparison of proposed design and reference designs will be given.

Part II
Design of a Typical Entropy Coder

Chapter 4
Design of a CABAC Encoder

In this chapter, architecture of a typical hardware (HW) CABAC encoder is illustrated. The design methodology of a SoC (System-on-Chip) based entropy coder is first presented. Based on the design methodology, hardware/software (HW/SW) functional partitioning of CABAC encoder function is carried out to decide which functions are to be designed in HW and which functions are to be processed on host processor. Furthermore, the strategy of functional partitioning of HW encoder is applied, and the top-level function blocks and encoding flow of the encoder are introduced. In the subsequent sections, design details of major functional blocks are presented including binarization and bin packet generation and binary arithmetic coding (BAC). Finally, additional functions supported by the encoder including context model initialization, RDO function support in BAC are discussed.

4.1 Design Methodology for the SoC-Based Entropy Coder

Design methodology for a SoC-based entropy coder such as CABAC is proposed [Le06], in which the entropy coder is realized as an IP block and can be integrated into a SoC video coding system. The IP block can be a logic synthesizable RTL design or a hard IP block where physical implementation is performed, and GDS-II file is ready. The design flow, as shown in Fig. 4.1, contains 9 steps including: performance and complexity analyses, derivation of system specifications, HW/SW functional partitioning, HW top-level functional partitioning, function block design of HW IP, HW IP verification, applying constraints and circuits synthesis, introducing SoC features, and HW/SW co-simulation.

S1. Performance and complexity analyses: It is critical to assess the complexity of software to be mapped onto the supposedly application specific top-level architecture. When the complexity is low, decision can be made to run the software using the existing host processor. If the complexity is high, it is better to identify the bottlenecks.
S2. System specifications are written in S2 to address not only user's needs but also plans to minimize the effects of bottlenecks.

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6_4,

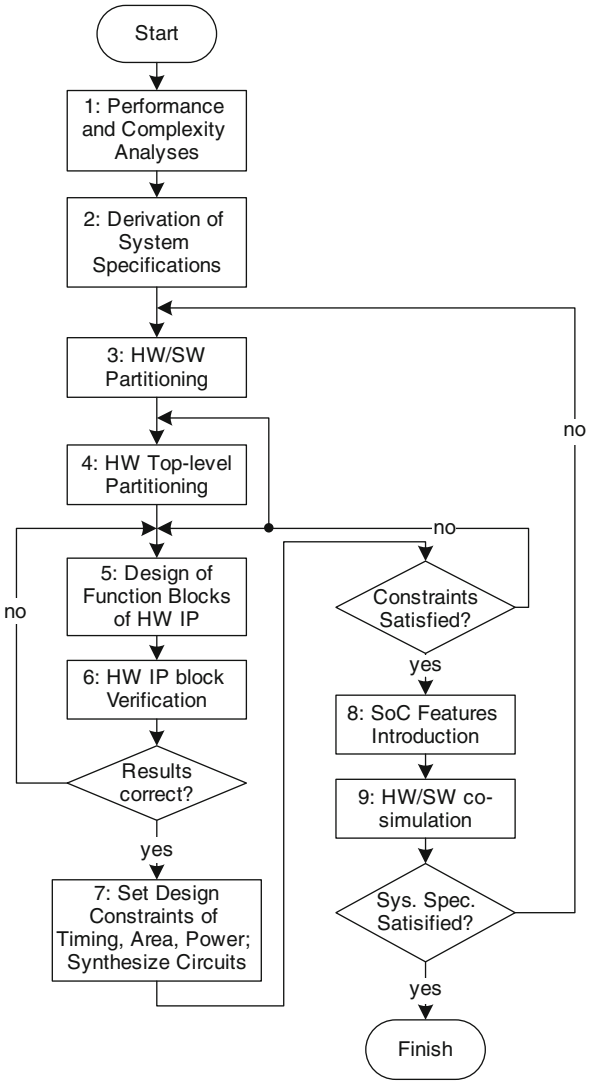

Fig. 4.1 Design flow for an SoC-based entropy coder

S3. System HW/SW partition is carried out: The portion of coding function – which can be effectively realized into HW – is replaced by HW preferred modules, while the remaining portion is left to be run in SW on the existing host processor. The decision by which HW/SW are partitioned is based on specifications such as coding speed of the entropy coder, and constraints of communication latency between host processor and the HW IP block, data transfer bandwidth on system bus, and remaining computation on the host processor. All SW preferred modules are referred to as one single SW non-IP block, while all HW preferred modules are referred to as one single HW IP block.

S4. The function of the HW IP block is further partitioned into proper functional blocks during top-level HW architecture design, which is crucial to the performance of HW IP.
S5. All functional blocks of HW IP are designed accordingly in step 5, followed by top-level HW IP block verification in step 6.
S6. Verification is performed by comparing the compressed bit stream generated by the reference SW (without any HW assisted circuitries), with that output by the top-level HW architecture. It is important to be certain that design errors are caught at this step.
S7. The RTL-level functional correct design is constrained with timing, area, and power restrictions and synthesized into gate-level circuits in step 7. If any of the constraints are violated, functional blocks are redesigned in step 5. If design violations still exist after critical functional block redesign, the design flow will go back to step 4 to adjust the top-level entropy coder architecture using a refined functional partitioning scheme. Design constraints can also be adjusted with tighter or looser constraints according to the synthesis results. The recursive procedure stops when all design constraints are met and the design is verified at gate-level.
S8. In step 8, SoC design features will be designed into the HW IP block. Such features are system bus interfaces and the signals of system bus, input and output buffers, debug structures, reset signal, etc. Function correctness and design constraints also need to be checked after SoC related function blocks integrated in step 8.
S9. Co-simulation of both HW IP block and SW non-IP block is done in step 9 to verify whether the system specifications of step 2 are satisfied, such as minimizing communication between SW non-IP block and HW IP block including data transfer latency and system bus bandwidth occupation.

4.1.1 Performance and Complexity Analysis of CABAC Encoder

The design methodology of the SoC-based entropy coder is applied to CABAC encoder design of H.264/AVC. Design step of performance and complexity analyses is presented first, and the following steps will be discussed in the later sections and chapters.

The coding performance of CABAC encoder IP design is compared to that of CAVLC (including Exp-Golomb coding for non-Residual SEs) by testing on H.264/AVC reference software JM 12.4 [WSJM] in the QP (quantization parameter) range of 28–40 using three video sequences in both CIF and HDTV 720p formats. The 3 sequences, named Seq1, Seq2, and Seq3 are: *Foreman*, *Coastguard*, and *News* in CIF format, and *City*, *Night*, and *Crew* in HDTV 720p format. Table 4.1 illustrates H.264/AVC encoder bit rate reduction, using CABAC compared to that with CAVLC.

The table shows that CABAC achieves an average bit rate reduction of 10.0% in CIF test and 15.5% in 720p test over CAVLC. The benefit is more significant in high

Table 4.1 H.264/AVC encoder bit rate reduction, using CABAC compared to with CAVLC

Format \ Sequence	RDO mode	Seq1 (%)	Seq2 (%)	Seq3 (%)	Average (%)
CIF	off	9.0	13.8	6.7	9.8
	on	10.1	13.7	6.8	10.2
HDTV 720p	off	12.4	11.1	22.4	15.3
	on	13.3	10.6	23.4	15.7

definition sequence and low bit rate (high QP) range. The performance difference is similar in RDO-off (RDO not used) mode and RDO-on (RDO used) mode.

Instruction-level program analyzing tool PIN [WSPIN] is used for profiling reference JM encoder. Compared to CAVLC and Exp-Golomb, the computational complexity of CABAC is higher by up to 55%, while its data transfer rate is higher by up to 74%, when RDO is used [Ho06]. This is partly due to the additional instructions required for binarization and context modeling (context model selection and access from memory). High computation and bandwidth make it difficult for CABAC reference SW to meet requirements of real-time encoding. It is, therefore, necessary to accelerate CABAC encoding by ASIC design.

In order to assist decisions in the following design steps including HW/SW functional partitioning and HW top-level functional partitioning, instruction-level complexity analysis is carried out on the CABAC encoder using JM reference software. All CABAC related functions are classified into 5 categories, from C1 to C5. The details of CABAC functions of each category are shown in Table 4.2.

PIN tool is utilized to profile the number of instructions of each category of CABAC encoding functions during H.264/AVC video coding. The percentage of instruction numbers of each category of CIF sequence test is calculated and listed in Fig. 4.2 and Table 4.3.

From Fig. 4.2 and Table 4.3, it can be seen that CA and BAC (C1) only occupies 32.9–45.2% of total CABAC instructions, and the percentage number decreases

Table 4.2 Five categories of CABAC encoder for instruction-level analysis

Category number	Description of operations
C1	Context access (CA) and Binary arithmetic coding (BAC)
C2	Non-residual SE coding, excluding computation of CA and BAC
C3	CBF coding, excluding computation of CA and BAC
C4	Coding of residual SEs including SCF, LSCF, and level, excluding computation of CA and BAC
C5	Operation of mapping quantized block coefficients to run-level pairs by scanning of coefficient block
C2 and C3	Coding of non-residual SE and CBF, excluding computation in CA and BAC (more complex for context model selection)

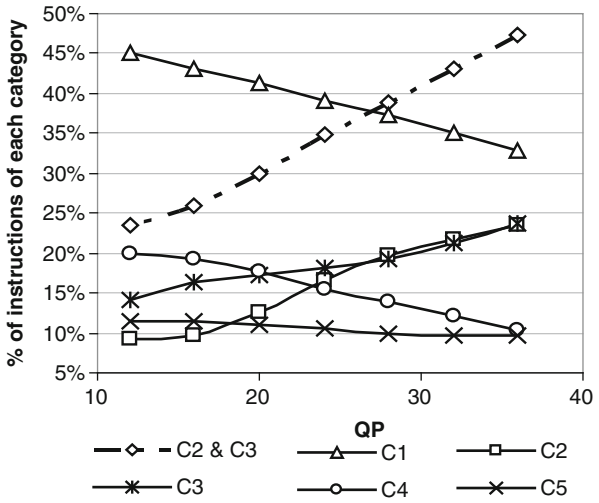

Fig. 4.2 Five CABAC functional categories as % of total CABAC instructions in CIF test of H.264/AVC encoder of JM reference SW in the QP range of 12–36

Table 4.3 Percentage of instructions of each category of the CABAC encoding functions in CIF sequence analysis

Category / QP	C1 (%)	C2 (%)	C3 (%)	C4 (%)	C5 (%)	C2 and C3 (%)
36	32.85	23.50	23.75	10.30	9.61	47.24
32	35.07	21.72	21.32	12.17	9.72	43.04
28	37.34	19.73	19.18	13.84	9.91	38.91
24	39.16	16.56	18.25	15.56	10.47	34.80
20	41.40	12.59	17.26	17.63	11.12	29.85
16	43.19	9.71	16.34	19.21	11.55	26.05
12	45.17	9.33	14.18	19.92	11.41	23.51
Average	39.17	16.16	18.61	15.52	10.54	34.77

when QP increases. CBF coding excluding computation of CA and BAC (C3) occupies 14.2–23.8% of total CABAC instructions, because of high data traffic and complex operation of context model selection (*CtxIdx* calculation) of CBF. The remaining computation (C4) of SCF, LSCF, and level, excluding CA and BAC occupies 10.3–19.9% of total CABAC instructions. The mapping procedure of run-level pairs to 4×4 block coefficients (C5) takes up 9.6–10.5% total CABAC instructions. The sum of % instructions of C2 and C3 is shown as dashed line in Fig. 4.2, representing the sum of non-residual SE and CBF coding (excluding CA and BAC). This sum occupies 23.5–47.2% of total CABAC instructions, and increases when QP increases. The instruction-level analysis is also done in HDTV 720p test, with analyzing result shown in.

Table 4.4 Percentage of instructions of each category of the CABAC encoding functions in HDTV 720p sequence analysis

Category / QP	C1 (%)	C2 (%)	C3 (%)	C4 (%)	C5 (%)	C2 and C3 (%)
36	32.41	21.30	27.04	9.17	10.08	48.34
32	33.27	19.01	26.19	11.03	10.50	45.20
28	34.49	16.54	24.87	13.22	10.89	41.41
24	35.81	12.73	24.25	15.67	11.53	36.98
20	38.25	8.76	22.56	18.28	12.16	31.31
16	41.47	7.86	18.72	19.72	12.22	26.58
12	44.35	9.45	14.69	19.85	11.66	24.14
Average	37.15	13.66	22.62	15.28	11.29	36.28

In Table 4.4, compared to that of CIF sequence coding, percentage of C3 increases significantly, while that of C1 decreases. This is because details of HDTV video are smoother, and the probability is higher that all coefficients of residual block are quantized to zero, and the instruction ratio of CBF coding increases, and that of other residual SEs decreases. Average of C2 and C3 instructions increases from 34.8 (CIF) to 36.3% (HDTV 720p).

4.1.2 Derivation of System Specifications for CABAC Encoder

The second step of SoC-based entropy coder design is the derivation of system specifications. This is performed before HW/SW functional partitioning, and detail targets of the design are specified. As CABAC is adopted in the Main Profile and High Profiles targeting high bit-rate high resolution H.264/AVC encoding, the CABAC encoder IP design is required to achieve real-time HDTV encoding of resolution of 720p and 1080p at 30 fps or higher frame rate, support different video coding configurations including adaptive frame/field (AFF) coding mode selection and RDO, minimize remaining computation on the host processor, minimize bandwidth and transfer delay on the system bus, and constrain circuit area and power consumption of the design. Based on these general system specifications, different HW/SW functional partitioning schemes are evaluated to decide which of the 5 CABAC encoding functional categories aforementioned and which encoding configurations should be accelerated by the encoder IP.

4.2 HW/SW Functional Partitioning of CABAC Encoder

HW/SW functional partitioning is the precondition of an efficient HW IP design of a particular function block of SoC-based video encoder, in which function of HW IP and SW non-IP on host processor, and data communication manner between host

processor and HW IP are decided. Based on the system specifications derived in step 2, different HW/SW partitioning schemes of CABAC encoder are analyzed and evaluated. Furthermore, necessity of supporting RDO functions in CABAC encoder IP is analyzed and emphasized.

4.2.1 Analysis of Different Partitioning Schemes

In the 3 main CABAC encoding steps binarization (BN), context modeling (CM), and BAC, CM can be further partitioned into context model selection (CS) and context model access (CA), and CABAC encoding can sequentially be executed through steps of BN, CS, CA, and BAC. CS can be further grouped into two categories: CS_1, if coded SEs of neighboring MBs on the left or top of current block of current MB are needed during context model selection; or CS_2, if this type of information is not needed. Five possible schemes of HW/SW functional partitioning of CABAC encoding in the reported designs and typical design reported in this book are analyzed subsequently. In Fig. 4.3, the functions partitioned to HW implementation of each scheme are highlighted as blocks, while the functions remained in SW are shown as "SW".

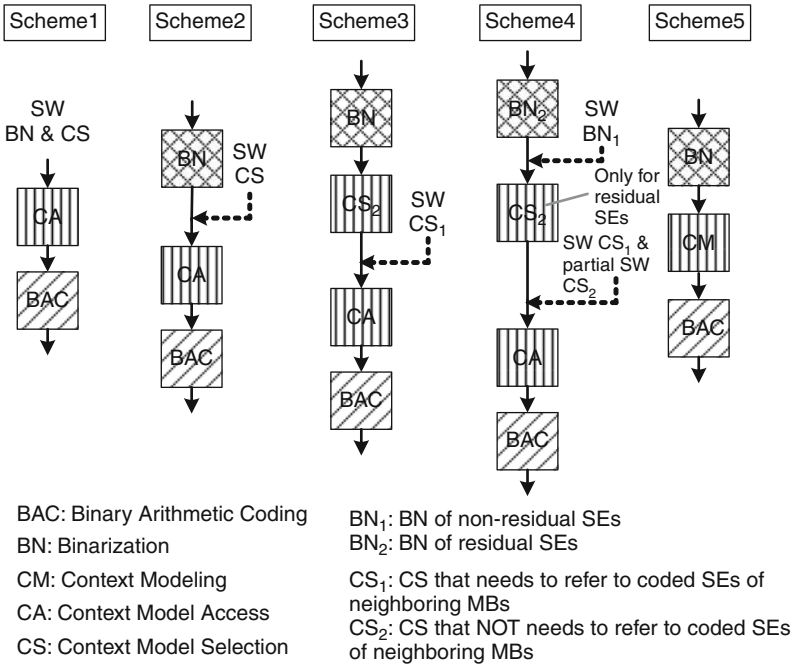

Fig. 4.3 Five HW/SW partitioning schemes of CABAC encoding

4.2.1.1 Scheme 1: HW Design of CA and BAC

As shown in Fig. 4.3, the HW partition only contains CA and BAC, while the remaining functions of BN and CS are to be run on the host processor. This scheme is implemented by several designs including [Kuo06, Li06a]. In some designs [Li06b, Nunez06], CA is left as SW, and both binarized bin string of SEs and corresponding context models are required to be prepared by SW, and input to HW IP through HW/SW interfaces. As shown in Sect. 4.1.1, CA and BAC occupies only 39.2% of the total CABAC instructions, leaving 60.8% of the remaining instructions to be run on the host processor. System bus bandwidth is also high because the value of coding bin and context model index (*CtxIdx*) of each regular bin need to be transferred from host through system bus. In general, CABAC encoding acceleration is insignificant in scheme 1.

4.2.1.2 Scheme 2: HW Design of BN, CA, and BAC

In this scheme [Sudharsanan05], binarization is also performed in HW. Only context model selection is left to SW. However, binarization still needs to be executed in SW because for a number of SE types, bin values of previous binarized bins are used for context model selection of following bins during encoding the SE. Computation resources of HW and SW are overlapped in this scheme, and CABAC is the same as that of scheme 1.

4.2.1.3 Scheme 3: SW Implementation of Context Model Selection of CS_1

In this scheme, only when coded SEs of neighboring blocks/MBs are referred to select context model (calculation of *CtxIdxInc*), the CS function is partitioned to SW (shown as CS_1 in Fig. 4.3). This scheme is adopted in our previous work [Tian08], because compared to the scheme 1 and scheme 2, large ratio of CS operations are removed from the host processor. CS_1 calculation is only necessary for some types of non-residual SEs and CBF of residual block, and the calculation is only for the first bin of SE, while for the remaining bins, CS_2 is adopted instead, which is implemented in HW. In addition, binarization is completely removed from SW non-IP block in this scheme, and computation on the host processor is further reduced. However, calculation of CS_2 is complex and inefficient in SW. Profiling results of Sect. 4.1.1 illustrate that computation of C2 and C3 occupies over 1/3 of CABAC instructions in average, and large ratio of computation is utilized for CS_2. For instance, CS_2 of CBF utilizes 18.6–22.6% of CABAC instructions. SW design of CS_2 also reduces integratability of the encoder IP and increases bandwidth on system bus.

4.2.1.4 Scheme 4: Scheme 1 with HW Support for Residual SEs of BN and CS, Except CBF

In this scheme [Osorio05, Osorio06], the design of CABAC HW encoder focuses on accelerating coding of residual SEs including SCF, LSCF, and coefficient level.

Binarization and CS of non-residual SEs are all partitioned to SW. Although the ratio of bins of SCF, LSCF, and level is large with respect to the total number of encoding bins, the remaining computation of SW is still significant (23.5–47.2% of CABAC instructions in CIF test, and 24.1–48.1% in HDTV 720p test in QP range of 12–36). Consequently, CABAC encoding can not be significantly accelerated, as the processing speed is restrained by the SW non-IP.

4.2.1.5 Scheme 5: Complete HW Implementation of CABAC Encoder

As aforementioned, computational burden of the host processor can be significantly reduced only when binarization and context model selection (CS) of non-residual SEs and CBF are implemented in HW. In scheme 5, the functions of BN, CS, CA, and BAC are all assigned to HW, and the remaining computation burden for SW is data packet preparation for the HW encoder, and coding result receiving from HW encoder. Compared to the other schemes, computation of host processor and data transfer on the system bus are minimized. Although the complexity of HW encoder is higher to support CS_2, and larger memory is required to store the coded SEs utilized by CS_2, CABAC encoding is significantly accelerated since the bottleneck of at SW non-IP is removed. In addition, integratability of HW IP of CABAC encoder is also enhanced. Considering the benefits of scheme 5, the proposed CABAC encoder IP of this chapter is based on this HW/SW functional partitioning scheme.

4.2.2 Analysis on the Supporting RDO Function in HW CABAC Encoder

To analyze the influence of RDO on the performance of H.264/AVC encoder, encoding simulation of JM reference encoder is carried out using same test sequences of Table 4.1, for both RDO-on mode and RDO-off mode. As shown in Table 4.5, RDO achieves similar bit-rate reduction ratio in the test when two entropy coders CABAC and CAVLC are used. For CABAC encoding, RDO contributes an average bit rate reduction of 11.8% in CIF test, and 16.4% in HDTV 720p test. The test results of Tables 4.1 and 4.5 indicate that it is necessary to support RDO in H.264/AVC encoder to obtain the significant benefit of the combinational coding gain of CABAC and RDO.

Computational complexity of CABAC encoder significantly increases in RDO-on mode. As shown in Table 4.6, the ratio of instructions of CABAC encoding in the CIF format to the H.264/AVC of RDO-off mode is 0.17–1.2%, while the ratio is 6.6–20.6% in RDO-on mode. Support for RDO in the CABAC encoder is necessary to accelerate CABAC encoding in RDO-on mode.

During RDO-on coding, CABAC encoder is utilized to calculate and feedback coding rate (length of the coded bit stream) of each RDO mode, and operations of coding state backup and restoration are required when RDO mode changes. The coding state of CABAC encoder includes three aspects: (a) the state of all context

Table 4.5 Bit rate reduction of H.264/AVC encoder, using RDO-on mode compared to RDO-off mode

Coder \ Sequence	Format	Seq1 (%)	Seq2 (%)	Seq3 (%)	Average (%)
CAVLC	CIF	11.1	13.0	10.2	11.4
	HDTV 720p	14.0	16.3	16.9	15.7
CABAC	CIF	12.2	13.0	10.3	11.8
	HDTV 720p	14.9	15.9	18.5	16.4

Table 4.6 Computational complexity of CABAC encoder in RDO-off/RDO-on mode

RDO mode	QP	Instructions of H.264/AVC	Instructions of CABAC	Ratio of CABAC/H.264 (%)
RDO-off	28	1.87E+10	3.13E+07	0.17
	24	1.88E+10	4.89E+07	0.26
	20	1.88E+10	8.05E+07	0.43
	16	1.89E+10	1.43E+08	0.75
	12	1.89E+10	2.23E+08	1.18
RDO-on	28	2.38E+10	1.58E+09	6.64
	24	2.45E+10	2.10E+09	8.56
	20	2.57E+10	2.99E+09	11.64
	16	2.76E+10	4.45E+09	16.09
	12	2.96E+10	6.10E+09	20.63

models of context modeling (CM); (b) state of coding interval of Range and Low of BAC; and (c) the state of coded SEs which are referred by CS_1 during context model selection. These coding state backup and restoration operations are necessary during RDO-on coding because of data dependency during arithmetic coding. In order to support RDO in H.264/AVC, it is required to support (a) the coding rate accumulation and feedback of each RDO mode, and (b) the coding state backup and restoration in CABAC encoder. These two types of RDO operations are partitioned to HW IP in this chapter to enhance top-level CABAC encoding performance by removing the bandwidth of huge amount of data transfer of RDO coding states between HW IP and SW non-IP, significantly longer memory access delay caused by access of both system memory and local embedded memory of HW IP, and more complex control logic for the RDO related data transfer.

4.3 HW Encoder Functional Partitioning Schemes, and the Top-Level Encoder Architecture

FSM-based functional partitioning scheme is adopted in the earlier design stage of proposed HW encoder. CABAC coding steps of binarization (BN), context modeling (CM), and BAC (Fig. 4.4a) can be implemented as a *Finite State Machine*

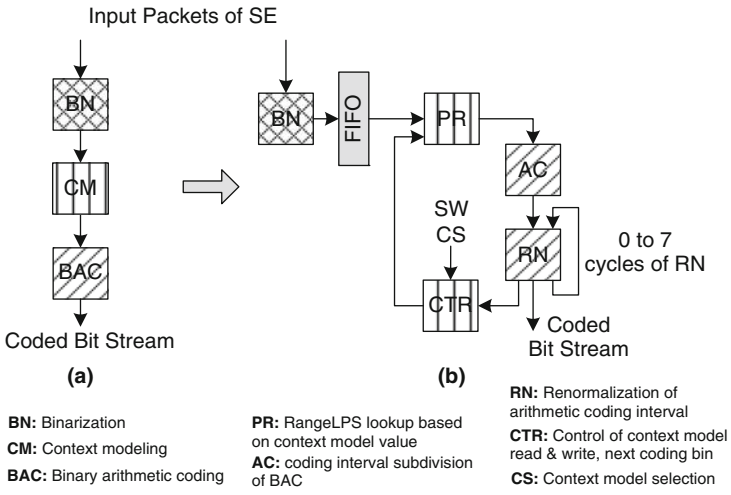

Fig. 4.4 FSM-based partitioning scheme for HW CABAC encoder

(FSM) (Fig. 4.4b). BN is separated from CM and BAC, with a FIFO buffer inserted between the two parts of functions. In BN, SE is binarized to bin string and packed with additional parameters including SE type, bin string length, etc. and buffered in the FIFO before being processed by the encoding FSM of CM and BAC.

The FSM consists of four sequential coding steps (states): PR ($Range_{LPS}$ look-up based on the context model value), AC, RN (renormalization), and CTR (control of context model), with function of CM mapped to states PR and CTR, and that of BAC mapped to AC and RN. In PR, $Range_{LPS}$ (Range value of LPS) of each regular bin is prepared by a table lookup based on context model and bin value. In AC, coding interval of [Low, Low+Range) is subdivided and updated by the selected sub-interval. In RN, updated coding interval is renormalized to maintain precision of arithmetic coding, while coded bits are output during renormalization. In the control state CTR, the encoding bin is next prepared with the corresponding context model of regular bin read from context RAM based on context model. RAM write operation of updated context model is executed in state RN during interval renormalization. RAM read and write are scheduled in different states as single-port context RAM is adopted. Compared to regular bin coding, bypass bin coding only goes through states AC and CTR, as context model access is not required.

Encoder coding throughput of FSM-based partitioning scheme is low because it takes 2 or more cycles to encode one bin (average of 4.3 cycles for each RB and 2 cycles for BB). The throughput is not constant because renormalization of state RN, which is based on the reference algorithm of the standard [ISO/IEC 14496-10], takes variable numbers of cycles from 0 to 7, with an average of 1.3 cycles. It is not easy to enhance CABAC encoding throughput, because of variable cycles required for renormalization in state RN.

4.3.1 Typical Hardware Functional Partitioning Scheme

The analyses of HW/SW functional partitioning schemes aforementioned prove that it is necessary to support complete CABAC function of BN, CM, and BAC in HW IP. FSM-based HW functional partitioning scheme is inefficient, because of sequential bin encoding manner. The proposed partitioning scheme targets acceleration of CABAC encoding by exploiting parallelism in different bin encoding steps.

4.3.1.1 Preliminary Partitioning

A preliminary stage of the proposed HW functional partitioning is shown in Fig. 4.5b. At this stage, the three coding steps of CABAC algorithm shown in Fig. 4.5a are partitioned into function units of BN, CS_1, CS_2, CA, and BAC. CM is partitioned to two consecutive coding stages CS and CA for context model selection and context model access. CS is partitioned to two units CS_1 and CS_2 based on whether coded SEs of neighboring MBs needed to be referenced during selection. The CABAC algorithm is executed in sequential order from BN, CS, CA, to BAC. The reason that CS and CA are allocated as two sequential coding stages after BN is that: (a) in some conditions, CS of current bin depends on the value of previous bin of same bin string; (b) for non-binarized SE, CS and CA are multi-cycle procedure that loop over all regular bins of the SE, while BN can be completed in single cycle, and it is beneficial to separate BN and CM to enable parallel processing of BN and CM; (c) CA depends on the context model selection result of CS, and it must be

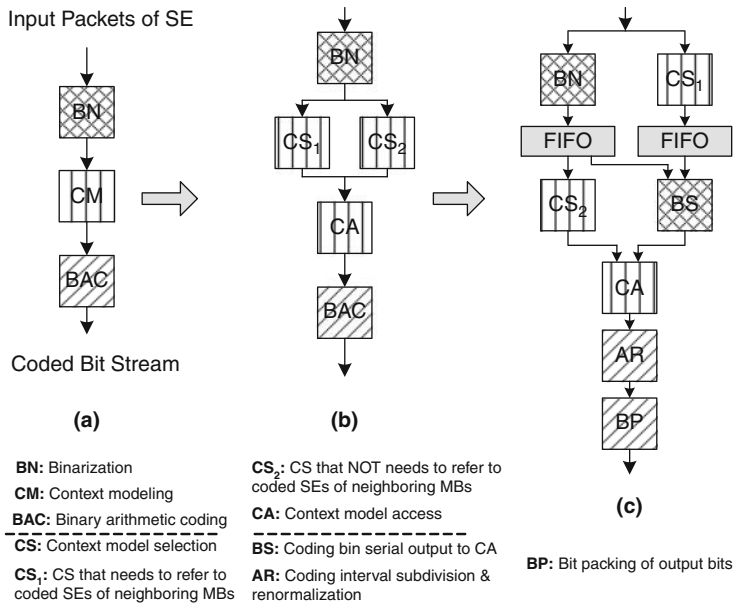

Fig. 4.5 The proposed partitioning scheme for HW CABAC encoder

scheduled after CS. After the preliminary partitioning, further functional partitioning and adjustment are necessary to target high-throughput fully pipelined encoding. Data dependency among the coding stages should be reduced, and the computational complexity among the partitioned functional units should be balanced.

4.3.1.2 Detailed Functional Partitioning Scheme for SE Coding

The proposed HW functional partition for SE coding is illustrated in Fig. 4.5c. Compared to the preliminary partitioning (Fig. 4.5b), unit BS (coding bin serial output to CA) is allocated to fetch bin string of SE generated by BN and serially generate packet of coding bins of bin string to CA and BAC. Additional parameters including bin type and *CtxIdx* of regular bin are packed with coding bin in single packet, named *bin packet* in BS, which will be processed by CA and BAC. In order to reduce the critical path length of BAC, the BAC is further partitioned into two coding units, including: unit AR for interval subdivision and renormalization of arithmetic coding, and unit BP for bit packing of output bit stream.

CS_1 is scheduled in the coding stage before CS_2 in CABAC encoding flow. The reason is that compared to CS_2, computation of CS_1 is more complex and irregular, and the selection of context model of CS_1 needs to reference and access the coded SEs of neighboring blocks/MBs. Moreover, as a further 9-bit addition of *CtxIdxInc* and *CtxOffset* is required to calculate *CtxIdx* of each regular bin, the critical path length can be efficiently reduced when the complex *CtxIdxInc* calculation is performed in CS_1, and *CtxIdx* is calculated and packed into bin packet in CS_2. Therefore, CS_1 is scheduled as the previous coding stage of CS_2 in the CABAC encoding flow, processing each input SE packet in parallel with BN. Although BN and CS_1 utilize similar input SE packet parsing circuit, packet processing mechanisms of the two are distinctly different. The two units are not combined in this scheme to avoid increase of circuit complexity and critical path delay and enhance coding efficiency. Because CS_1 is only triggered to calculate *CtxIdxInc* of the first regular bin of several types of SEs, it is ensured in this scheme that *CtxIdxInc* selected in unit CS_1 is ready for *CtxIdx* calculation in unit BS when the bin packet of corresponding regular bin is to be generated in BS. Partitioning of CS functions of this scheme has no influence on the throughput of bin packet generation of BS.

Two FIFO buffers are inserted to buffer the output of unit BN and CS_1: bin string packet of BN and *CtxIdxInc* of CS_1. FIFO buffer insertion enables BN and CS_1 to work in parallel with the following coding units. FIFO buffer insertion after BN is necessary because for the coefficients of residual block, it takes 1–16 cycles to receive input packets of run-level data pair before bin string generation of residual SEs including CBF, SCF, LSCF, and level. During this output idle period of BN, the buffered bin strings in FIFO can still provide bin strings to unit BS without interruption.

The function units partitioned in the proposed scheme (Fig. 4.5c) include N and CS_1, BS and CS_2, CA, AR, and BP. The units are scheduled as sequential SE coding stages. Data dependency among these coding stages is minimized, and the units consist of a top-level SE encoding pipeline, in which all units work in parallel.

Compared to the FSM-based partitioning scheme aforementioned, 1 bin/cycle throughput can be achieved in the proposed scheme. For regular bin coding, the throughput speedup of the proposed scheme compared to FSM-based scheme is 4.3; and for bypass bin coding, the speedup is 2. The top-level full-pipelined encoder architecture will be introduced in the following sections.

4.3.1.3 Functional Partitioning of Additional Functions of CABAC Encoder

As discussed in Sect. 4.2, additional functions need to be supported in the HW partition of CABAC encoder, including (a) initialization of context models of context RAM during slice initialization and (b) context state backup and restoration during P8×8 sub-MB RDO mode decision. Because these two types of functions are not frequently triggered during CABAC encoding compared to SE coding and context memory access operations are required in both, the two types of functions are partitioned to one function block in the proposed CABAC encoder. The block and the other SE encoding units of the encoder are activated alternately. When the block is triggered by the control signals parsed from input packets, the other SE encoding units of the encoder will stay in idle state until operation of the block completes. HW acceleration of this block is necessary to reduce idle time of the top-level SE coding pipeline. Compared to the reported designs, design of this function block is only proposed in this work.

4.3.2 Fully Pipelined Top-Level HW CABAC Encoder Architecture

The top-level architecture of the proposed CABAC encoder is shown in Fig. 4.6. The encoder consists of three functional blocks and a memory block: Block 1 for input parameter parsing and binarization and context model selection (CS); Block 2 for CA and BAC; Block 3 for functions of context model initialization and context state backup and restoration during P8×8 sub-MB RDO mode decision (P8×8 RDO coding); and a memory block for context RAMs and ROM tables.

As shown in the figure, Block 1 consists of 3 functional units CS_1, BN, and BS&CS_2. It also utilizes 3 FIFO buffers and a RAM of coded SEs. The RAM is only accessed by CS_1. The function of BS and CS_2 are integrated into one unit (BS&CS_2) in order to complete the context model selection (*CtxIdx* calculation) and packing of bin and *CtxIdx* in the same cycle. Block 2 consists of 3 functional units including CA, AR, and BP, as introduced in Sect. 4.3.1. The WISHBONE (WB) system bus master and slave interfaces [Wishbone] are also integrated to the encoder to enhance the portability and reusability of the IP, and design of system bus interfaces will be discussed in Chap. 6.

Input SE received from WB slave interface is encoded in Block 1 and Block 2. In Block 1, FIFO1 (16-word×22-bit) buffers 22-bit input SE packets. Unit BN binarizes SE value into 33-bit bin string package. Read of FIFO1 is controlled by BN. The packet read from FIFO1 is sent to and parsed in BN and CS_1 simultaneously. Context models of part of regular bins are selected in CS_1 by calculating a 3-bit

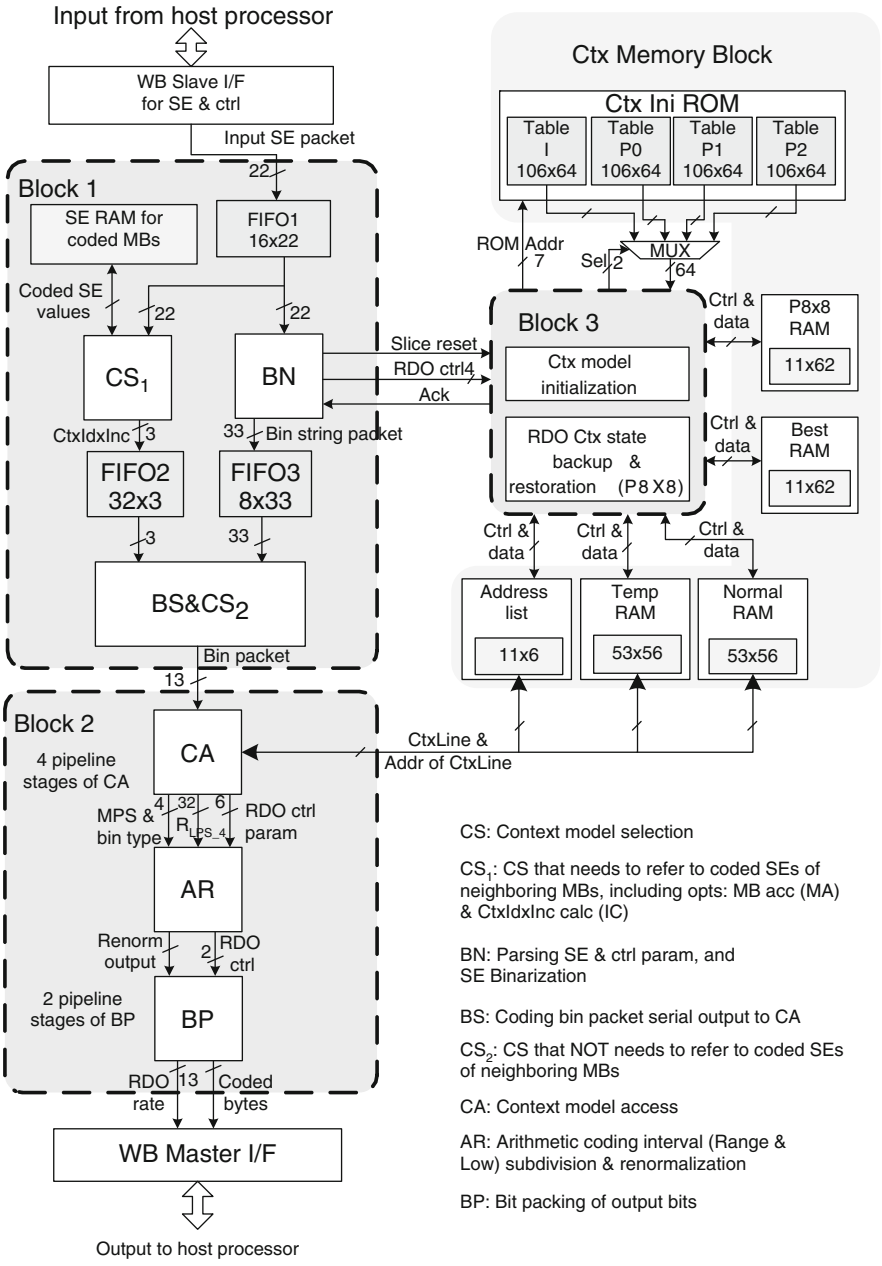

Fig. 4.6 Block diagram of top-level architecture of HW CABAC encoder

CtxIdxInc when the coded SEs of neighboring blocks/MBs are to be referenced, which are stored in the SE RAM. The size of SE RAM depends on the maximum horizontal resolution of input video sequences supported by the encoder. In unit BS&CS$_2$, context model selection of regular bin is completed by calculating *CtxIdx* of each regular bin based on SE type and calculated *CtxIdxInc* from CS$_1$ or CS$_2$. For each bin of SE bin string, a 13-bit bin packet containing bin value, bin type, and *CtxIdx* of regular bin is generated in BS&CS$_2$, and sent to Block 2. As previously discussed, a 32-word×3-bit FIFO2 and a 8-word×33-bit FIFO3 are inserted in Block 1 to buffer the bin string of BN and *CtxIdxInc* of CS$_1$ that can largely reduce the probability of stalling in unit BS&CS$_2$ and the following coding stages when the input data to BS&CS$_2$ is not available. In Block 2, context model of each regular bin is accessed in unit CA from context RAMs according to *CtxIdx*. The coding interval is subdivided and renormalized in unit AR by updating values of Range and Low. After renormalization, upper bits of Low are parsed in unit BP, and packed and output through the WB system bus master interface. Coding throughput of top-level pipeline of Block 1 and Block 2 is 1 bin/cycle, as in each cycle one bin packet can be generated in BS&CS$_2$ and encoded in AR.

The context memory block of the encoder (top-right of Fig. 4.6) contains one context initialization ROM (4 ROM tables, 106-word×64-bit each) used by Block 3, and five context RAMs including a 53-word×56-bit Normal context RAM (Normal RAM), a 53-word×56-bit Temp context RAM (Temp RAM), a 11-word×6-bit context line access address list (Address list), a 11-word×62-bit RAM that stores coding address and context line in P8×8 RDO coding (P8×8 RAM), and a 11-word×62-bit RAM that stores address and context line in P8×8 RDO coding (Best RAM). Normal RAM, Temp RAM, and Address list are accessed by both Block 2 and Block 3, while P8×8 RAM and Best RAM are only accessed by Block 3. In the following sections of this chapter, unit BN and BS&CS$_2$ of Block 1 and unit AR and BP of Block 2 will be discussed in details. Context model selection and access including CS$_1$ and CA will be discussed in Chap. 5.

4.4 Binarization and Generation of Bin Packet

4.4.1 Input SE Parsing and Binarization of Unit BN

Input packets to the CABAC encoder provide the necessary information for both SE coding and encoder control. The packets are buffered in FIFO1 and accessed by both units BN and CS$_1$. The format of the 22-bit packet is illustrated in Fig. 4.7. The 2 highest significant bits (21:20) decide one of 4 categories of input packets: non-residual SE, Run-Level pair of residual block coefficients, RDO coding control parameter, and slice initialization parameters and frame/filed coding flag at the frame level. For non-residual SEs, 4-bit tag (19:16) is used to classify among different SE types including MVD-S (absolute MVD value that can be represented in 6 bits), MVD-L (large MVD value), reference index, MB QP delta, Skip flag,

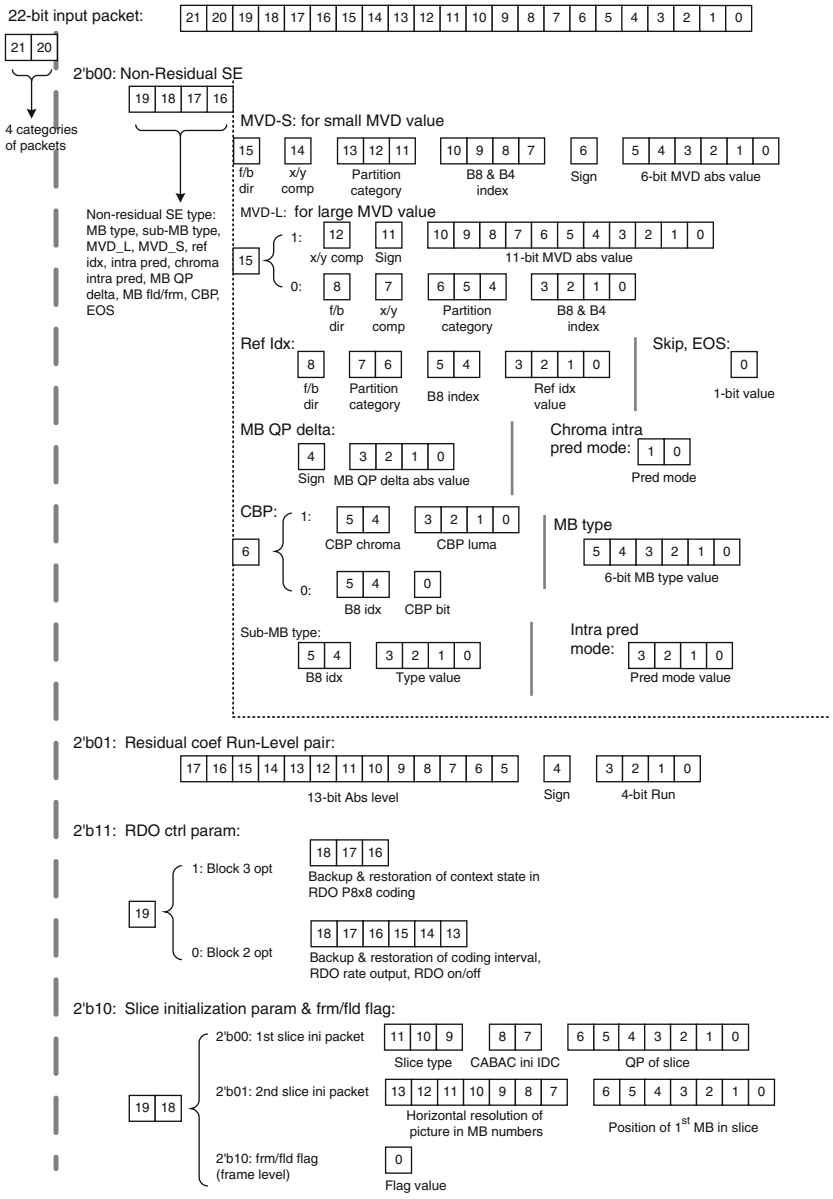

Fig. 4.7 Input packet format of CABAC encoder

EOS (end of slice), luma/chroma intra prediction mode, CBP, MB type, and sub-MB type.

For residual coefficients, the packet of run-level pair supports the 13-bit absolute value of level, and 4-bit run, which are the maximum range of the Main profile.

Two types of RDO control parameters are supported that control RDO operation of context state backup and restoration in Block 3 and coding interval backup and restoration and coding rate output of each RDO mode in Block 2. Slice initialization parameters are parsed from the two input packets including slice type, slice QP, CABAC initialization IDC, the number of MBs in the horizontal resolution of picture, and the position of the 1st MB of slice. The slice initialization parameters are utilized in BN and CS_1 and Block 3.

Details of input packet parsing and SE binarization procedure of unit BN are illustrated in Fig. 4.8. BN contains 3 major data paths that process input packets of non-residual SEs, residual SEs, and the encoder control parameters. When the control parameters are parsed and sent to Block 3, BN waits for the acknowledgment from Block 3 before processing the next input packet. The End-of-Slice (EOS) flag and RDO operation parameters including Range and Low backup and restoration modes, RDO on/off flag, and RDO coding rate output instruction are buffered in the FIFO3 and forwarded to units of Block 2.

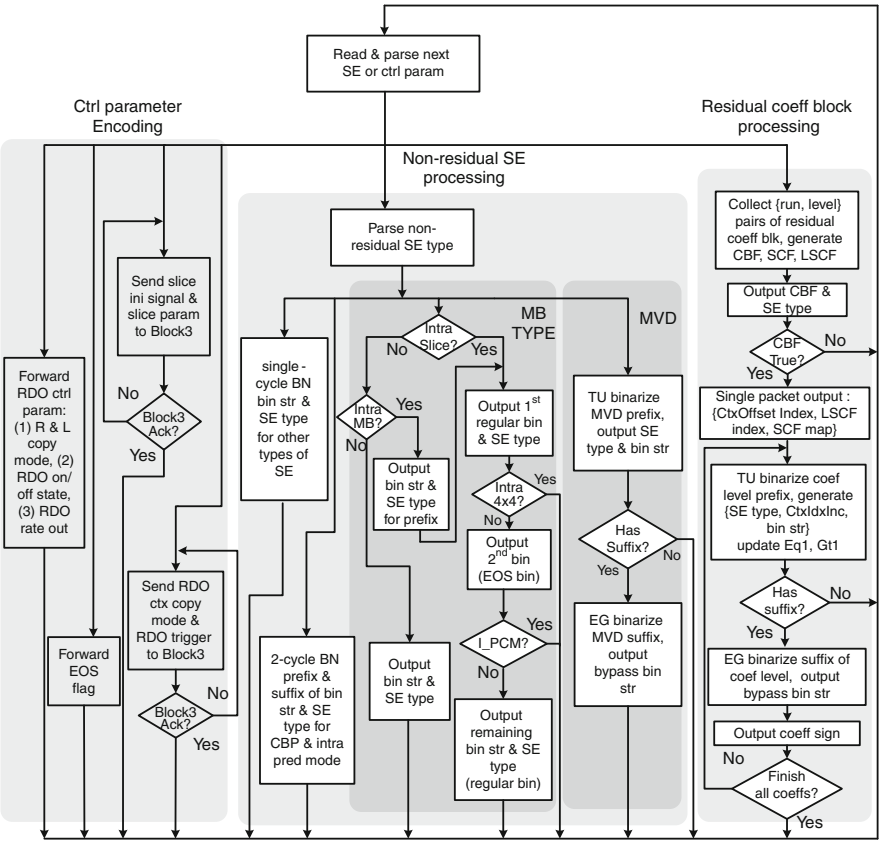

Fig. 4.8 Procedure for parsing and binarization non-/residual SE and control parameters of unit BN, Block 1

Different types of non-residual SEs shown in Fig. 4.8 are parsed according to the 4-bit tag in the packet and binarized in BN. Except for the SE type and SE value that are utilized for binarization, additional parameters are provided in the input SE packets of some types of SEs including MVD, reference index, CBP, sub-MB type, etc. These parameters include forward/backward directions, partition category, and block index of 8×8 sub-MB and 4×4 block which provide the necessary information to locate and access coded SEs in the current MB, or neighboring coded MBs during context model selection of CS_1. Because several numbers of parameters of MVD are required in CS_1, in order to reduce packet size, input parameters of large MVD (abs MVD >=64) are separated to two packets, as shown in Fig. 4.8.

The throughput of CABAC encoder is not influenced by packet partitioning, because the frequency of encoding large MVD is very low, and it takes multiple cycles to generate bin packets of SE in BS&CS_2. In order to simplify packet parsing in BS&CS_1, prefix and suffix bin strings of SE are sent to BS&CS_1 in separate packets, and bin strings of regular bin, bypass bin, and EOS bin are also packed separately. A barrel shifter is utilized to implement binarization schemes of unary and truncated unary (TU) for SEs such as reference index, MB QP delta, prefix of MVD, and residual coefficient level. For the SEs with LUT-based binarization scheme, including MB type and sub-MB type of I, P, and B pictures, LUT is implemented by combinational circuits instead of memory-based LUT to reduce table lookup delay.

Coefficients of residual block are processed in different paths, because SE values are not available until run-level pairs of a 4×4 block are all received. Figure 4.8 shows that SEs of various flags, such as CBF (coded block flag), SCF (significant coefficient flag), and LSCF (last SCF) are generated during run-level pair receiving procedure. After the output of CBF, a single packet of SCF and LSCF are output, which contains 15-bit SCF map, 4-bit index of LSCF position, and 4-bit *CtxOffset* index consisting of frame/field coding flag and residual block category. The benefits of single output packet of SCF and LSCF include: total processing time of residual block in unit BN is reduced, and the coding throughput of BS&CS_2 is significantly improved. The absolute level values (*abs_level*) and signs of coefficients are output in LIFO (last in first out) pattern. *CtxIdxInc* of the first bin and the remaining bins of abs_level prefix bin string are calculated in BN and sent to BS&CS_2 according to the number of coded level values equal to 1 (Eq1), and number of coded levels of value greater than 1 (Gt1) of the processing block, as shown in Fig. 4.8. The counters of Eq1 and Gt1 are accumulated for each block during coefficient level binarization procedure.

For SEs of MVD (motion vector difference) and residual coefficient levels, binarization techniques of TU (truncated unary) and EGk (kth order Exp-Golomb code) are used to binarize the prefix and suffix bin strings, respectively. The EGk binarization introduced in Chap. 2 is not suitable for HW implementation because of long and variable operational delay. A fast EGk coding circuit is proposed and applied in unit BN for suffix binarization of MVD (k=3) and coefficient level (k=0).

```
Assume input value of EGk is x, with m+1 bits, represented by
x[M:0],and k >= 0
For k=0:
      y=x[M:0]+1, MSB of y is y[N], N can be M or M+1
      The output bin string contains 2N+1 bits, including N bits of
      1, 1 bit of 0, and lower N bits of y: y[N-1:0]
For k>0:
      x[M:0] is the concatenation of two parts:
      x[M:k] and x[k-1:0]
      y=x[M:k]+1, MSB of y is y[N], N can be M-k or M-k+1
      The output bin string contains 2N+1+k bits, including
      N bits of 1, 1 bit of 0, y[N-1:0], and x[k-1:0]
```

Fig. 4.9 HW-oriented EGk binarization algorithm

In general, the HW-oriented EGk binarization algorithm can be described in the following pseudo code, shown in Fig. 4.9.

According to the algorithm depicted by Fig. 4.9, EG3 circuits that support 11-bit range of absolute MVD is designed to generate a MVD suffix bin string in single cycle, as shown in Fig. 4.10a. The absolute value of MVD is subtracted by 1 first (equal to: abs_MVD – 9 + (1 ≪ *k*), in which k is 3). The 8 most significant bits of abs_MVD-1 are checked in the detection circuit of Most-Significant-One (MSB), and bin N is detected as Most-Significant-One. The output suffix string consists of N bits 1, one bit 0, and lower N+3 bits of abs_MVD-1. MSB index and the sign of MVD are output in the same packet. A fast EG0 circuit similar to that of MVD is designed for the abs_level suffix binarization, which supports maximum range of bin string length of 25, as shown in Fig. 4.10b.

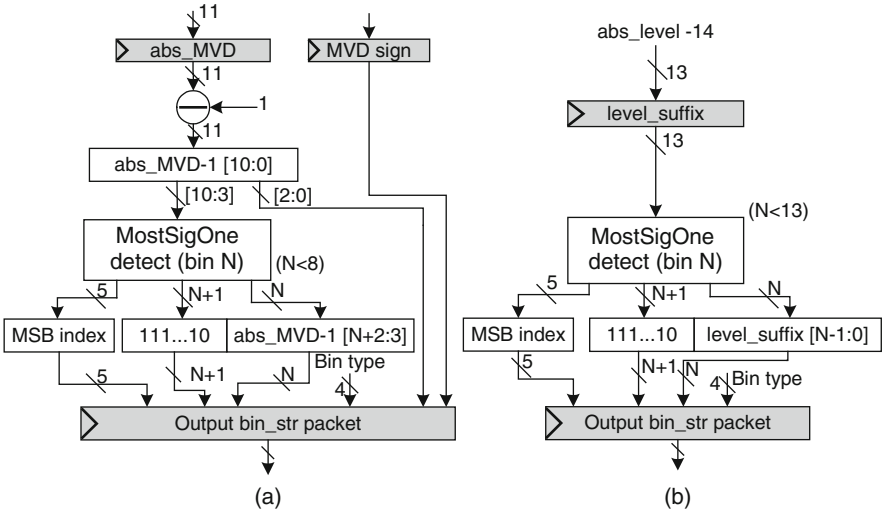

Fig. 4.10 Fast EGk binarization implementaion. (**a**) EG3 binarization for the suffix of MVD; (**b**) EG0 binarization for the suffix of abs_level_minus1

4.4.2 Bin Packet Generation and Serial Output of Unit BS&CS$_2$

Unit BS&CS$_2$ fetches 33-bit bin string packet from FIFO3 and serially outputs bin packet for each bin of SE bin string with the related context information to unit CA of Block 2. For regular bin, context model selection is completed in this unit by calculating *CtxIdx*, which is used to locate the context model of coding bin from context RAM in unit CA. For regular bin, the 13-bit bin packet consists of bin value, *CtxIdx*, and bin type. For bypass bin, terminate bin (including EOS), and control parameters of Block 2, no calculation of *CtxIdx* is needed. The architecture of unit BS&CS$_2$ is shown in Fig. 4.11.

Figure 4.11a illustrates processing procedure of regular and bypass bin string excluding SCF and LSCF of residual block. Regular or bypass bin in the string is output from MSB to LSB. *CtxIdx* value of regular bin is calculated by the addition of *CtxOffset* and *CtxIdxInc*. *CtxOffset* decides the initial RAM address of a group of context models of the same SE type, while *CtxIdxInc* decides the position of the selected context model within the group. *CtxOffset* is looked up from LUTs according to SE type, which is packed to the bin string packet by unit BN. For large ratio of regular bins of non-residual SEs, *CtxIdxInc* is looked up from LUTs in unit BS&CS$_2$ based on bin index (position of the bin in bin string) and/or the value of previous bin in the bin string. Only for a small part of regular bins of non-residual SEs, *CtxIdxInc* is input from the context model selection results of unit CS$_1$, which are buffered in FIFO2; while for abs_level, context model selection is generated in unit BN, and *CtxIdxInc* values of the 1st bin and remaining bins are packed in the same input bin string packet of abs_level.

Bin packet generation of SCF and LSCF is shown in Fig. 4.11b. The map of SCF of residual block is stored in a 15-bit buffer, and bin packets of SCF and LSCF are output from the position of SCF index 0. The *LSCF/SCF coding flag* decides

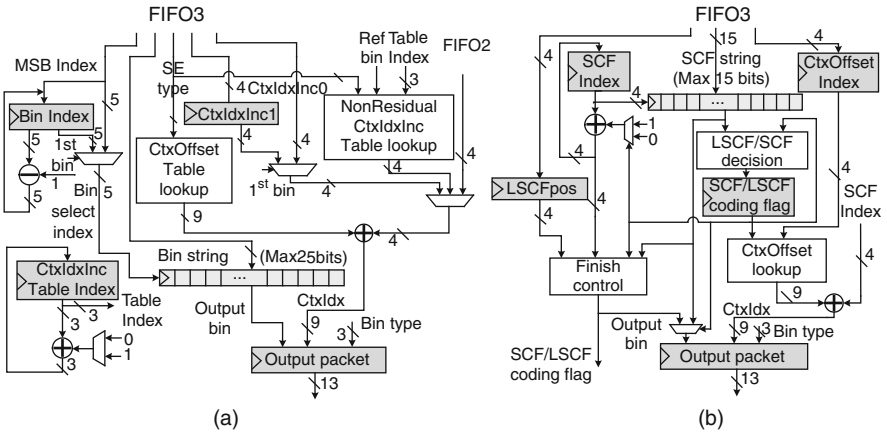

Fig. 4.11 Architecture of unit BS&CS2: (**a**) CtxIdx calculation and bin packet serial output circuit for all SE, excluding SCF and LSCF; (**b**) CtxIdx calculation and SE serial output of SCF and LSCF packet of residual coefficient block

the output order of bin packet of SCF and LSCF, and controls the increase of SCF index. Bin value of SCF in the packet is read from SCF map, while LSCF value is generated in the *Finish Control* block according to LSCF position. *CtxOffset* values of SCF and LSCF are looked up according to *CtxOffset* index, and *CtxIdxInc* is generated from SCF index directly. Context model selection of unit CS_1 will be discussed in Chap. 5.

The processing of SE bin string and generation of bin packet in unit BS&CS_2 are designed as FSM. According to bin type, SE type, and whether context model is selected by CS_1 or CS_2, first FSM state parses bin string and processes the first bin of string, and the other FSM states can be triggered to process following bins in the string. For each bin string packet, 1 bin packet can be generated per cycle. Processing steps of input packet paring, *CtxIdx* calculation, and bin packet generation and output can be completed in the same cycle. If the next bin string packet is ready in FIFO3 during processing of current bin string, unit BS&CS_2 will continue generation and output of bin packet of next bin string with no pause. Therefore, constant throughput of 1 bin packet/cycle is insured for the proposed design of unit BS&CS_2.

4.5 Binary Arithmetic Coding (BAC)

Binary arithmetic coding (BAC) is the encoding and bit stream output stage of CABAC. Functions of BAC include: (a) encoding of regular bin, bypass bin, and terminate bin by subdivision and selection of coding interval according to the context model (probability model) of coding bin, (b) renormalization of coding interval to keep encoding precision by upscaling Range and Low, and (c) output confirmed higher bits of Low as coded bits of CABAC. Proper functional partitioning and efficient removing of data dependency are important to accelerate BAC. As aforementioed, BAC is partitioned into two consecutive coding stages: AR and BP. In the proposed HW encoder, BAC is partitioned and designed as a 3-stage pipeline: first stage for unit AR and following 2 stages for unit BP. In the following sections, proposed HW-oriented renormalization and bit packing scheme is introduced first. Then, design consideration and architectures of the pipeline stages of BAC will be discussed respectively.

4.5.1 A Typical Renormalization and Bit Packing Algorithm

As discussed in the FSM-based functional partitioning scheme, direct implementation of SW-oriented renormalization algorithm is not timing-efficient and restricts the throughput of BAC. A HW-oriented renormalization and bit packing algorithm is discussed in this chapter, which provides identical coding performance, but significantly higher coding throughput, compared to the reference algorithm of H.264/AVC [ISO/IEC 14496-10]. The proposed algorithm, as described in the flowchart shown in Fig. 4.12, which is different from the QM coder renormalization

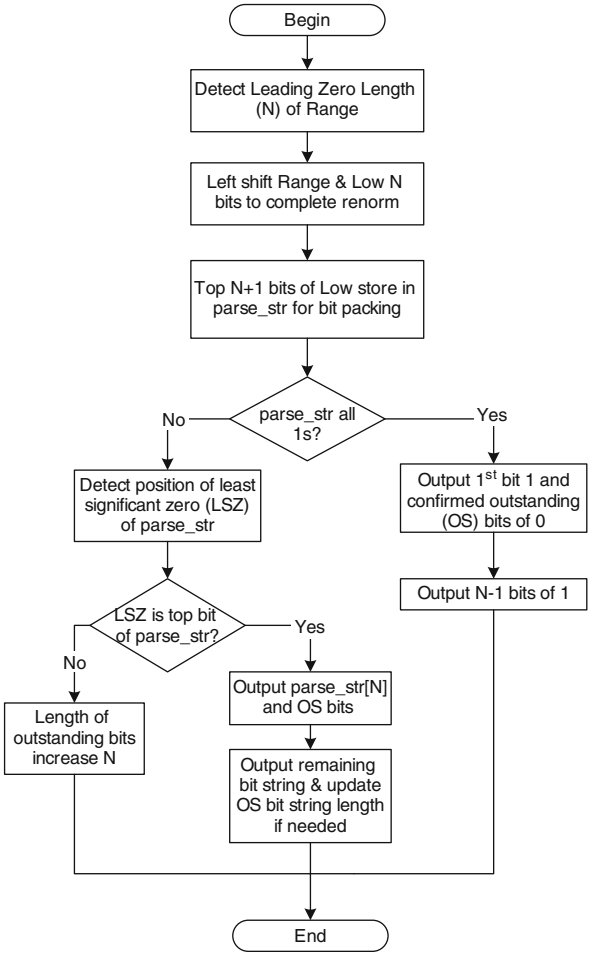

Fig. 4.12 Flowchart of renormalization and bit packing algorithm in HW

derived in [Nunez06, Osorio06]. After coding interval subdivision, Range and Low of selected subinterval are input to the renormalization and bit packing stage.

The proposed algorithm illustrated in Fig. 4.12 can be designed as two sequential steps for renormalization and bit packing. Renormalization can be completed within single clock cycle by checking leading zeros of updated Range, while bit packing is more complex. Critical path of bit packing can be reduced by implementing bit packing into two pipeline cycles, because there is no data dependency (feedback loop) in sequential bit packing operations. Leading zeros detection and least significant zero detection are implemented by fast table lookup implemented in combinational circuits, while the other operations are implemented as arithmetic operations including addition, barrel shift, bit concatenation, comparison, etc. Because outstanding bits are not output until bit value is confirmed, the issue of carry propagation is not involved in this design compared to that of [Nunez06, Osorio06].

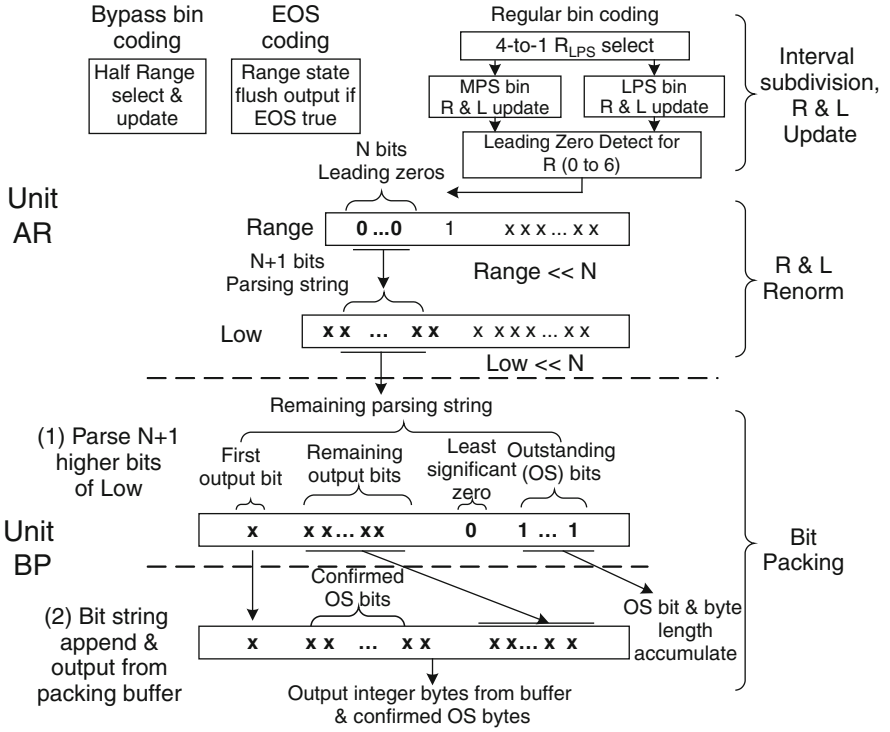

Fig. 4.13 Three-stage pipeline implementation of renormalization and bit packing algorithm in unit AR and unit BP

The proposed renormalization and bit packing algorithm is implemented as a 3-stage pipeline illustrated in Fig. 4.13. Because the updated Range after renormalization is required for the selection of proper $Range_{LPS}$ for the next regular bin, operations of coding interval subdivision and renormalization of Range and Low need to be completed within one clock cycle. Therefore, these operations are designed as sequential steps of unit AR of Block 2. Prefetch of $Range_{LPS}$ is necessary before AR operations, because the lookup of 64-entry LUT based on the 6-bit pStateIdx of context model is timing consuming. The prefetch of $Range_{LPS}$ is implemented during context model access of unit CA, which is one stage before the operation of AR, and it can efficiently reduce critical path length of unit AR.

Because Range value is not available during prefetch, 4 possible values of $Range_{LPS}$ of the next regular bin are all prefetched. As shown in Fig. 4.13, a 4-to-1 multiplexer is allocated in unit AR to select the correct $Range_{LPS}$ before coding interval subdivision based on 2 bits of updated value of Range (Range [7:6]). In unit BP, string parsing of higher bits of Low and the following operation of bit string packing and output are separated to two pipeline stages. The throughput is not influenced in continuous BAC coding procedure, while critical path of unit BP is significantly reduced.

4.5.2 Coding Interval Subdivision and Renormalization of Unit AR

The architecture of unit AR is shown in Fig. 4.14. Regular bin, bypass bin, and terminate bin (EOS) are encoded in 3 separate coding routes. The coding interval represented by Range and Low is subdivided and updated to one of the two subdivisions according to bin value, bin type, and MPS of bin in regular bin coding. For regular bin coding, interval subdivision and renormalization can be implemented as two sequential steps. Coding interval is updated to one of two subintervals based on whether bin is MPS or LPS. Then, Range and Low of the interval are renormalized.

In unit AR, MPS bin and LPS bin are processed separately, as shown in Fig. 4.14. For MPS bin, renormalization is not needed or only 1 bit renormalization is taken; while for LPS bin, length (1–6 bits) of renormalization is decided by a 5-bit leading zero detection (LZD) circuit of Range. Separate processing of MPS and LPS simplifies coding procedure and reduces critical path length in unit AR, compared to the scheme with a unified renormalization step for both MPS and LPS bin. Renormalization is executed by left shifting N bits (LZD length) of the updated Range and Low. Multiplexers are allocated to select proper results from one of the three coding routes including Range, Low, and coding parameters to the bit packing stage. The critical path of unit AR is from $\text{Range}_{\text{LPS}}$ selection to coding interval update and renormalization of LPS bin, to the data input of Range registers. In

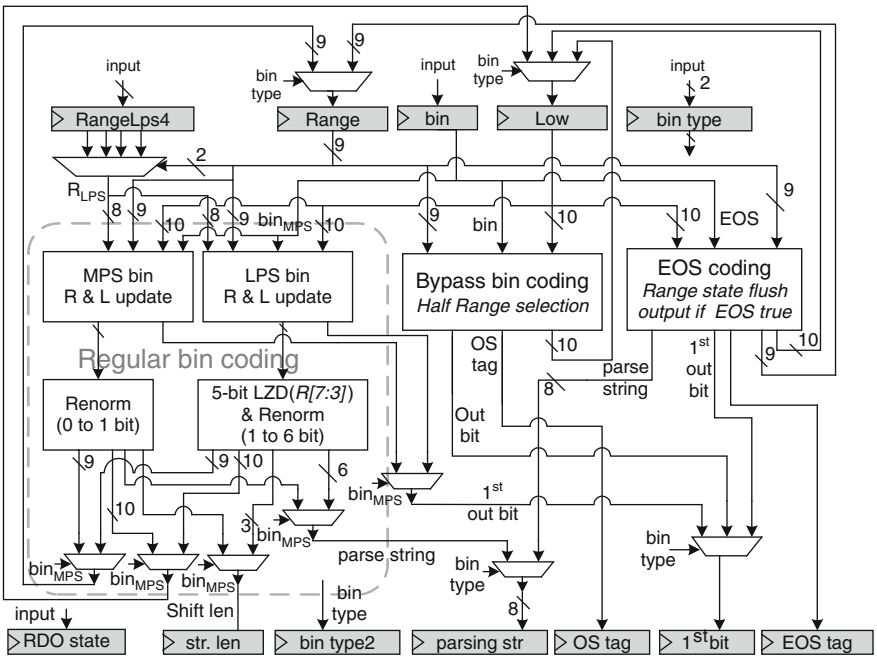

Fig. 4.14 Architecture of unit AR

[Osorio06], update and renormalization of Low is scheduled one cycle later than that of Range, because selection of $\text{Range}_{\text{LPS}}$ does not depend on the updated value of Low. However, because Low computation is not complex during interval subdivision and renormalization, it is beneficial to schedule the operations of Low and Range in the same cycle to avoid one cycle delay of bit packing.

4.5.3 Bit Packing of Unit BP

The early design version of this unit only contains single pipeline stage. In order to reduce critical path length of unit BP, a two-stage bit packing unit with 8-bit packet size is proposed, and the architecture is shown in Fig. 4.15. In the 1st pipeline stage of BP, least significant zero (LSZ) bit position of parsing string (the output bit string from Low after renormalization generated in unit AR) is determined. Based on LSZ position, parsing string is separated into output bit string and outstanding (OS) bit string. In the 2nd stage of BP, the first output bit, confirmed OS bits, and remaining output bits are appended in the packing buffer. Because output packet size is reduced from 32 to 8 bits (1 byte), the size of packing buffer and operation delay of bit packing are reduced significantly. A 3-bit OS byte counter and a 3-bit OS bit counter are allocated to calculate the accumulated length of OS bits, and the design can tolerate a maximum length of 63 bits of OS bit string (enough for extreme

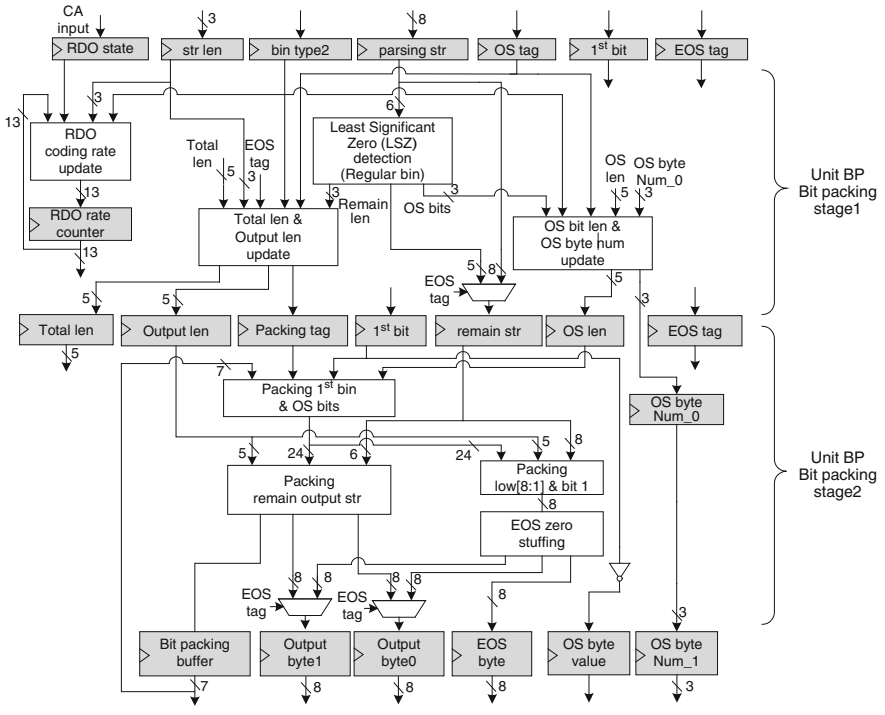

Fig. 4.15 Two-stage design of bit packing

case) before OS bit string value is confirmed. Coded bits are output from bit packing buffer in integer number of bytes with confirmed bytes of OS bits inserted after the 1st output byte.

As shown in Fig. 4.15, excluding the confirmed stuffing OS bytes, maximum of 3 bytes can be output from BP per cycle when EOS flag is coded in BAC, including EOS byte, 1st output byte, and 2nd output byte. Compared to CABAC encoder designs [Osorio04, Shojania05, Osorio06] with FIFO buffer allocated for output packets, output FIFO buffer is not inserted after unit BP, and output delay of coded packets is reduced. Situation of multiple output bytes in one cycle can be handled by the system bus interface of CABAC encoder, in which coded bytes are packed and serially output.

4.6 Additional Functions Supported by the CABAC Encoder

4.6.1 Context Model Initialization

Before CABAC coding of SEs of one slice, context models need to be initialized based on parameters of slice type, QP, and CABAC initialization IDC [ISO/IEC 14496-10]. Most of the reported designs do not implement context model initialization in HW. The advantage of HW context model initialization is that HW calculation is more timing-efficient compared to the calculation on the host processor, and system bus occupation for context model transfer and memory access of context initialization parameters is removed. As shown in Fig. 4.6, one of the 4 ROM tables is selected to provide context model initialization parameters to Block 3. Each ROM table is 106-line×64-bit, and each 64-bit line stores context initialization parameters of 4 context models. A 5-stage pipeline is designed to process 4 context models per cycle. Stage 1: read address of one line of ROM table is output from Block 3; stage 2: the ROM stores read address; stage 3: initialization parameters of 4 context models are read from ROM; stage 4: 4 multiplications are executed in parallel; and stage 5: 4 context models are generated in parallel, with one addition and several bitwise operations. Four identical processing units are allocated in this pipeline, and the calculation is further accelerated. With the additional computation resources allocated, idle cycles between coding slices are significantly reduced. In every two cycles, 8 context models are generated, concatenated, and written to Normal context RAM. A total of 424 context models can be initialized in 110 cycles, including pipeline filling time.

4.6.2 RDO Operations Implemented in BAC

In RDO-on mode coding, according to the RDO control parameters, state of coding interval (values of Range and Low) are backed up or restored in unit AR when RDO mode changes. Three backup copies of Range and Low values are allocated. During the coding of non-P8×8 RDO mode, only one copy of backup values are accessed, which stores the original state of interval; while in P8×8 RDO coding, 3 backup

copies are all utilized during selection of the best coding mode (partition mode of ME) of each 8×8 sub-MB.

The coding rate of each RDO mode is accumulated in the 13-bit counter allocated in the first pipeline stage of unit BP during coding of each RDO mode. Instead of parsing output bit string, length of code bit stream (number of confirmed output bits and outstanding bits) is accumulated in a local counter. When coding of one RDO mode is finished, the coding rate is output to the host processor through system bus interface. In SoC video encoder system of H.264/AVC with the proposed CABAC encoder, coding of the next RDO mode will begin only when the rate of current mode is received by higher level system controller (host processor). If there is no data dependency between consecutive coding modes, such as intra4×4 prediction mode, input coding packets of next mode can be sent to CABAC encoder after packets of current mode is sent, even the rate of current mode is not received by the host processor.

4.6.3 FWFT Internal FIFO Buffers

For conventional design of FIFO buffers, if FIFO *data-output-ready-signal* is set in cycle_N, FIFO read signal can only be sent to FIFO in cycle_{N+1}, and FIFO output data can only be used in cycle_{N+2}. One cycle of delay of FIFO read can not be avoided in such architecture. In the CABAC encoder design targeting high throughput, it is necessary that FIFO output data can be used in cycle_{N+1} when ready signal is set in cycle_N. One solution is to access FIFO at the negative edge of clock edge. However, clock frequency will be degraded because for the critical path start from FIFO output data, critical path is limited to half of cycle length.

In the proposed CABAC encoder design, FIFO of FWFT (first word fall through) architecture is designed and used for the 3 FIFOs in Block 1. The FIFO consists of a single port SRAM, combinational control circuits, and one output data buffer. The output ready signal is not registered so as to respond to the read request from Block 1 in the same cycle. Although circuit area of FWFT FIFO (with output data buffered) is larger than that of conventional FIFO, one cycle of FIFO read access delay is removed, and the throughput of CABAC encoder is not influenced by FIFO access.

4.7 Summary

In this chapter, Design Methodology of SoC-based Statistical Coder is presented first. Based on the methodology, performance and complexity analysis and system specifications of CABAC encoder are derived. According to the specifications, HW/SW functional partitioning and HW functional partitioning of CABAC encoder are carried out. A full-pipelined CABAC encoder is designed based on the partitioning schems, and the top-level architectures of the encoder and detailed architectures including binarization, BAC, etc. of Block 1 and Block 2 of the encoder are introduced, respectively.

Chapter 5
Efficient Architecture for Context Modeling in the CABAC Encoder

In this chapter, context-modeling-related functions of the proposed CABAC encoder design are introduced including context model selection of unit CS_1 and context model access of unit CA. A scheme of context line access and local buffering is proposed to reduce RAM access frequency. During each RAM access, multiple context models (8 models) are read from or written to context RAM instead of single context model access in most designs. Statistical data of CACTI RAM modeling tool [WSCACTI] demonstrates that context RAM power consumption of the proposed scheme can be lower when RAM access frequency reduces to 33.9% of the designs with single context model access architecture. With local context line buffering and elaborate context model reallocation of context RAM, efficient context model access can be achieved. Coding state backup and restoration operations of RDO-on mode are also introduced including the operations for context models of Block 3 and codes SEs of unit CS_1. Compared to the reported designs, full support of coding state backup and restoration of RDO-on mode is only proposed in this part of the book.

5.1 Context Model Selection

The outstanding compression efficiency achieved in CABAC encoder of H.264/AVC with comparison to the previous arithmetic coders can be attributed to the proper selection of context model and regular bin coding based on selected context model. For bypass bin coding, no compression can be gained. In order to select the most suitable context model of each regular bin, a number of factors need to be considered, including the SE type, the coded bins of current SE, the coded SEs of neighboring blocks, etc.

As introduced in Chap. 2, the context model of context RAM addressed by *CtxIdx*, which is the sum of *CtxOffset* and *CtxIdxInc*. The context model selection is partitioned into unit CS_1 and unit $BS\&CS_2$ of Block 1. In CS_1, the context model selection needs to reference to coded SEs of neighboring blocks of current MB or neighboring MBs. As shown in Fig. 5.1, unit CS_1 is further partitioned to two sub-units: IC sub-unit for coded SE access and *CtxIdxInc* calculation for current coding MBs, and MA sub-unit for memory access operations for the backup coded SE. The generated *CtxIdxInc* values of CS_1 are buffered in FIFO2 and utilized by $BS\&CS_1$

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6_5,

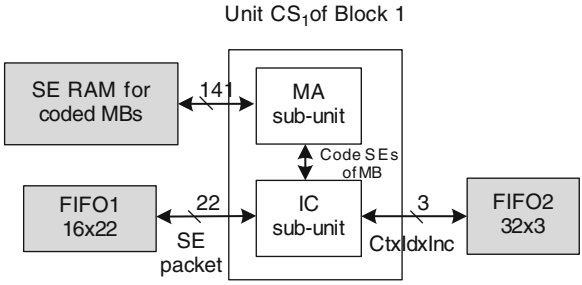

Fig. 5.1 Block diagram of unit CS1, with sub-units MA and IC

for *CtxIdx* calculation. In the following Sect. 5.1, the scheme for storage and fast access of the coded SE in IC sub-unit will be introduced first, followed by separate discussions of the architectures of IC and MA sub-units.

5.1.1 Scheme of Storage and Fast Access of Coded SEs of IC Sub-unit

IC sub-unit of unit CS_1 calculates *CtxIdxInc* for particular bins at specified position (bin index) of several types of SEs, as listed in Table 9.30 of [ISO/IEC 14496-10]. The coded SEs of neighboring BPMB (4×4 block, MB/sub-MB partition, or MB) on top or left of current BPMB are used for the calculation of *CtxIdxInc* of one regular bin of current SE in IC sub-unit.

The implementation of *CtxIdxInc* calculation with reference to the coded SEs in unit CS_1 is challenging because of the irregular access to the coded SE of neighboring BPMB for different types of SE. The SW-oriented derivation process of coded SEs of neighboring BPMB specified in the H.264/AVC standard [ISO/IEC 14496-10] can be described in the following steps:

S1. For the processed bin, the category of the SE is identified first whether the SE is of MB, partition, or block;
S2. The neighboring reference BPMB is derived in a complex procedure by first locating a neighboring pixel on the left or top of the top left pixel of current processed position, and then deciding which block, partition, or MB the located pixel belongs to;
S3. The SE of neighboring block, partition, or MB is accessed from the memory and used for *CtxIdxInc* calculation. The SW-oriented BPMB derivation is complicated and not suitable for HW design targeting fast access and high throughput. In order to complete the derivation of neighboring BPMB, access of coded SE of the BPMB, and the calculation of *CtxIdxInc* in single cycle with constrained critical path length, a scheme of storage and fast access of coded SEs of neighboring BPMB is proposed as follows. Figure 5.2 illustrates the basic concept of the scheme.

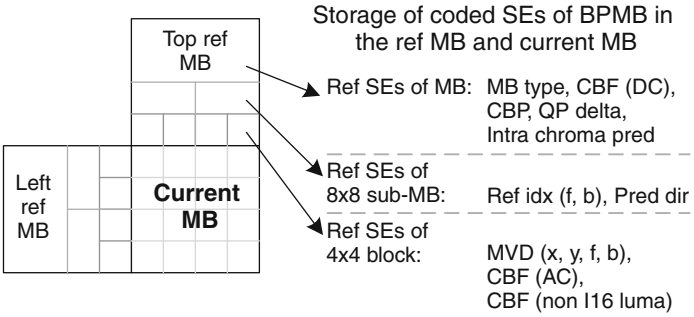

Fig. 5.2 Reference MBs on top and left of current MB, and storage of 3 categories of coded SEs (MB, 8×8 sub-MB, and 4×4 block) in the referenced BPMB of the current and referenced MBs

As shown in Fig. 5.2, the coded SEs of variable types of BPMB are stored in 3 levels: MB level, 8×8 sub-MB level, and 4×4 block level. The classification is based on the minimum size of BPMB of the coded SE:

- MB level: For the SEs including MB type, CBF of DC coefficient block, CBP, QP delta, intra chroma prediction mode, the minimum size is 16×16 and the SEs are classified to MB level and stored one SE per MB.
- Sub-MB level: For the SEs including reference index and prediction direction, the size of BPMB can be 16×16, 16×8, 8×16, and 8×8, based on the partitioning mode of MB. As the minimum size of BPMB is 8×8, the SE is stored for each 8×8 sub-MB partition of the BPMB that can be accessed in IC calculation of following SEs of same type. For example, for a BPMB of size 16×8, 2 copies of SE values need to be stored for the corresponding two 8×8 sub-MBs in the BPMB.
- Block level: For the SEs including MVD, CBF of AC coefficient block, and CBF of luma coefficient block of non-Intra16×16 MB, the minimum size of BPMB is 4×4, and coded SE values, are stored for the 4×4 blocks in the BPMB that can be further accessed in IC calculation. Although BPMB size of MB/sub-MB partition of MVD is in the range of 16×16 to 4×4, MVD is stored in the unit of 4×4 block.

In the reference MBs on the top and left of current MB, the SEs of MB level are all stored. For the SEs of sub-MB level and block level, only the coded SEs of the 4 blocks and 2 sub-MBs on the neighboring edge of current MB can be accessed and need to be stored. In the IC sub-unit of unit CS_1, the necessary SEs of 3 levels in the reference MBs are prepared before coding of current MB, while the coded SEs of current MB are stored for the MB, sub-MBs, or blocks based on which level the SE is classified to. Although more than one copy of SE value can be stored for the SEs of sub-MB level or block level including MVD, reference index, and prediction direction, access to the coded SE is much more efficient in circuit complexity and

processing speed (throughput × clock frequency) for HW design, compared to SW-oriented procedure aforementioned.

5.1.1.1 Fast Access of Coded SEs of Neighboring Block or Sub-MB

Fast access scheme of the neighboring coded block or sub-MB of the IC sub-unit is illustrated in Fig. 5.3. For block-level SE, a 4-bit block index of the SE is provided for input SE packet. The range of block index is from 0 to 15, with the higher 2 bits indicating one of the four 8×8 sub-MBs in the MB, and the lower 2 bits indicating one of four blocks in the sub-MB. The 4 blocks of left-referenced MB are indexed from 16 to 19, and the 4 blocks of top-referenced MB are indexed from 20 to 23. The derivation of block index of left and top block of current block is based on the combinational circuit implementation of LUT. The LUT of block index can be derived from Fig. 5.3a, as shown in Table 5.1. The 4-bit index of current block is mapped to 5-bit index of neighboring block in the range of 0 to 23, with MSB (most significant bit) indicating whether the block is in the current MB or neighboring coded MB.

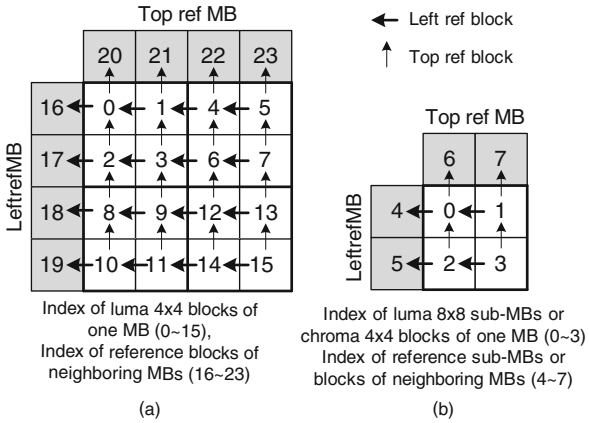

Fig. 5.3 Fast access of neighboring coded block and sub-MBs. (**a**) Access of neighboring luma 4×4 blocks, and (**b**) access of neighboring 8×8 sub-MBs and chroma 4×4 blocks of 4:2:0 sampling format

For the 8×8 sub-MBs and chroma 4×4 blocks of the 4:2:0 sampling format of the Main profile of H.264/AVC, the index of block or sub-MB of current MB is from 0 to 3, as shown in Fig. 5.3b. The two sub-MBs or blocks of the left-referenced MB of current MB are indexed with values 4 and 5, while the two of top-referenced MB are indexed with values 6 and 7. As shown in Table 5.2, the fast LUT of 3-bit index of neighboring sub-MB or block can be implemented based on the 2-bit index of the current MB. The LUT can be derived from Fig. 5.3b, and is implemented as the combinational circuit to reduce access time.

In one special situation of *CtxIdxInc* calculation, it is necessary to access coded SE of 8×8 sub-MB that contains the neighboring 4×4 block on the left or top of current 4×4 block. The sub-MB index can be derived in two steps:

Table 5.1 Fast table lookup of block index of neighboring block on the left or top of current block for block level SE processing

Current block index	Left block		Top block		Current block index	Left block		Top block	
	Index	Current MB	Index	Current MB		Index	Current MB	Index	Current MB
0	16	0	20	0	8	18	0	2	1
1	0	1	21	0	9	8	1	3	1
2	17	0	0	1	10	19	0	8	1
3	2	1	1	1	11	10	1	9	1
4	1	1	22	0	12	9	1	6	1
5	4	1	23	0	13	12	1	7	1
6	3	1	4	1	14	11	1	12	1
7	6	1	5	1	15	14	1	13	1

Table 5.2 Fast table lookup of block/sub-MB index of neighboring chroma block/8×8 sub-MB on the left or top of current block/8×8 sub-MB

Current block/sub-MB index	Left block/sub-MB		Top block/sub-MB	
	Index	Current MB	Index	Current MB
0	4	0	6	0
1	0	1	7	0
2	5	0	0	1
3	2	1	1	1

- Decide whether neighboring block locates in current sub-MB or neighboring sub-MB, based on the lower two bits of 4-bit block index of current block and left/top direction of neighboring block.
- If the block is in current sub-MB, the derived sub-MB index is 2 higher bits of block index of current block; Otherwise, the sub-MB index is looked up in the LUT (Table 5.2) based on 2 higher bits of block index of current block.

To accelerate access of coded SE in this situation, a fast LUT is established using 2 two-step derivation, and implemented as combinational circuit in IC sub-unit to accelerate sub-MB locating and SE access. The LUT is shown in Table 5.3, in which input to the LUT is the current block index, and output is sub-MB index of the neighboring block.

Block level and sub-MB level SEs of current MB and neighboring reference MBs are stored in registers of the blocks and sub-MBs indexed in Fig. 5.3a, b. During *CtxIdxInc* calculation of block/sub-MB level SE in unit CS_1, fast access of reference SEs of neighboring block/sub-MB is implemented through index lookup on the LUT and direct read of SE value from local registers, instead of complicated calculation and memory access of SW-oriented procedure.

Table 5.3 Fast table lookup of sub-MB index of neighboring block on the left or top of current block based on current block index

Current block index	Sub-MB index		Current block index	Sub-MB index	
	Left block	Top block		Left block	Top block
0	4	6	8	5	0
1	0	6	9	2	0
2	4	0	10	5	2
3	0	0	11	2	2
4	0	7	12	2	1
5	1	7	13	3	1
6	0	1	14	2	3
7	1	1	15	3	3

5.1.1.2 Storage of Coded SEs of Reference MB

The storage of codes SEs of the top- and left-referenced MBs are shown in Table 5.4. A total of 141 bits are allocated for top- or left-referenced MB to store all necessary coded SEs that can be accessed by IC sub-unit during *CtxIdxInc* calculation of current MB. QP delta of previous coded MB needs to be stored separately from SEs of top and left MB, and it is only accessed by current MB. In order to reduce memory storage and simplify computation in IC sub-unit, the values of several types of SEs are not stored directly, but remapped to a small set of values represented by fewer bits. MB type in the range of 0 to 48 is mapped to 3-bit value that classifies 7 types of MBs. Reference index, intra chroma prediction mode, and QP delta are represented by 1 bit, indicating if the value is zero or not. MVD values occupy large amount of storage of reference MBs and current MB, because for each 4×4 block, 4 MVD values need to be stored. Although absolute MVD value of 11 bits is supported in the proposed CABAC encoder, it is not necessary to store all 11 bits for each MVD Selection of context model is only influenced by MVD with small values. Therefore, 7 bits are allocated for each MVD instead of 11 of proposed scheme, with MSB indicating if absolute MVD is less than 64 or not. The benefit is that for each reference MB, 64 bits of SRAM storage is reduced, and the size of SE RAM reduces by 31.2%.

As shown in Table 5.4, for SEs of block level and sub-MB level including reference index, prediction direction, and MVD, the positions of block/sub-MB of coded SEs that are stored for top and left reference MBs are different, which is discriminated by the index of block/sub-MB in the table. The 2-bit prediction direction is utilized to indicate if one MB/sub-MB partition is of forward, backward, or bi-directional prediction in B slice. Although prediction direction is not coded in the later stages of CABAC, the value is referenced during *CtxIdxInc* calculation of MVD and reference index in IC sub-unit. Therefore, 2 values (2 bits each) are stored for 2 sub-MBs of each reference MB. In the proposed scheme, local storage of coded SEs of MB level and CBF values of DC coefficient blocks is shared for top

Table 5.4 Storage of coded SEs of top/left-referenced MBs

Stored SE type	Num of SEs	Stored bits/SE	Location of SE in MB, and usage description
MB type	1	3	3 bits to classify 7 types of MB in I, P, and B picture
CBP	1	6	6 bits for 4:2:0, 4 bits for luma 8×8 sub-MB, 2 bits for chroma 8×8 sub-MB
IntraChroma prediction	1	1	Indicate if intra chroma prediction mode is of DC mode or not
Reference index (f/b)	2×2: 4	1	Top: Forward/backward direction, 2 of 8×8 sub-MBs of index 2 and 3 at MB bottom, 1 bit indicates reference index is 0 or not
			Left: Sub-MBs w/index: 1 and 3
Prediction direction	2	2	Top: 2 of 8×8 sub-MBs w/index 2 and 3 at MB bottom
			Left: Sub-MB index: 1 and 3
MVD component (f/b and x/y)	4×4:16	7	Top: For the 4 of 4×4 blocks of index 10,11,14,15 at bottom of MB, 4 MVD components per block for forward/backward directions, x/y components
			Left: 4 blocks with index 5,7,13,15
CBF (DC)	3	1	1 for DC of luma of I16 MB, 2 for DC blocks of chroma (Cb and Cr)
CBF (AC & non-intra16×16 luma block)	8	1	Top: 4 bits for luma 4×4 blocks of index 10, 11, 14,15; 4 bits for chroma 4×4 blocks (2 of Cb and 2 of Cr of index 2 and 3) at MB bottom
			Left: Luma block w/ index: 5,7,13,15, and chroma block w/ index: 1, 3 on the right side
Total stored bits/MB		1×3+1×6+1×1+4×1+2×2+16×7+11×1 = 141 bits	
QP delta	1bit/MB, indicating if QP delta is 0 or not		

and left reference MBs, so that 13 bits of registers can be reduced in IC sub-unit. This part of SEs are stored in the local registers of left reference MB first, and then moved to SE RAM of top reference MBs.

5.1.2 CtxIdxInc *Calculation (IC) of Unit* CS_1

CtxIdxInc calculation (selection of context model) of unit CS_1 is implemented in the IC sub-unit, which parses each input packet of CABAC encoder, stores coded SE values in local register buffers, calculates the 3-bit *CtxIdxInc* with reference to the coded SEs of neighboring BPMBs, and writes *CtxIdxInc* values into FIFO2, which will be accessed by unit BS&CS_2 of Block 1. As shown in Fig. 4.7, additional

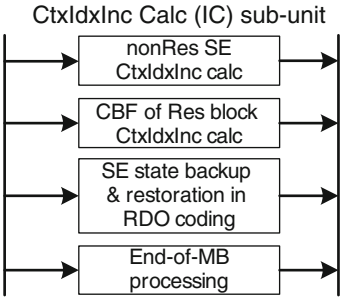

Fig. 5.4 Functions of IC sub-unit of unit CS1

parameters are provided in input packets to assist processing of block/sub-MB level SEs, including block/sub-MB index, partition category, block category, etc. that are used to locate SE in the current MB, and provide storage position of coded SEs.

Different processing routes of IC sub-unit are illustrated in Fig. 5.4, including *CtxIdxInc* calculation of several types of non-residual SE, *CtxIdxInc* calculation of residual block (CBF), backup and restoration of the state of coded SEs in RDO-on coding mode, and End-of-MB process triggered by MB end flag. RDO related operations will be discussed in 5.4. Multi-cycle *CtxIdxInc* calculation is proposed in [Liu07a], in which 4 cycles are required to access codes SEs of top- and left-referenced BPMBs and 1 more cycle to select context model. Compared to [Liu07a], single-cycle *CtxIdxInc* calculation is achieved in the IC sub-unit of the proposed CABAC encoder for all types of SEs, which avoids the situation of CABAC encoding pipeline stall caused by the access of coded SEs of neighboring reference BPMBs, and ensures constantly high throughput of CABAC encoding. The functions of *CtxIdxInc* calculation and storage of coded SE values of different SE types are discussed separately.

5.1.2.1 CtxIdxInc Calculation of Different SE Types

Input packets of CABAC encoder are buffered in FIFO1 of Block 1. Read access of FIFO1 is controlled by unit BN of Block 1, while each packet is also sent to and parsed in unit CS_1 when it is read out from FIFO1. Packets are classified to different types of SEs or control parameters according to packet identification bits.

During *CtxIdxInc* calculation of IC sub-unit, required parameters for different SE types are diversified, as listed in Table 5.5. For most types of SEs processed in IC sub-unit (excluding QP delta), coded SEs of the two neighboring reference BPMBs (on the top and left of current BPMB) need to be accessed from local SE buffers of IC sub-unit. For the neighboring coded SE values of block/sub-MB level, index of BPMB (block/sub-MB) is decided through table lookup according to the index of 8×8 sub-MB (B8 index) and/or 4×4 block (concatenation of B8 and B4 index) from input packets, as shown in Fig. 4.7. Three LUTs aforementioned are utilized to derive index of neighboring reference block/sub-MB for the access of reference SEs, with utilization of combinational circuits.

Table 5.5 Parameters of reference BPMBs required for *CtxIdxInc* calculation of different types of SEs

SE type	Description of required parameters for *CtxIdxInc* calculation	
MB type	Reference MBs: MB availability, MB type	
Skip	Reference MBs: MB availability, MB skip	
QP delta	Reference MB (previous MB): MB type, QP delta, CBP	
Intra chroma prediction mode	Reference MBs: MB availability, MB type, intra chroma prediction mode	
CBP	Luma (4 bins for non-P8×8 mode, 1 bin for sub-MB of P8×8 mode): reference sub-MBs: MB availability, MB type, CBP bit of sub-MB (Table 5.2) Chroma (2 bins): reference MBs: MB availability, MB type, CBP chroma value	
MVD	Reference blocks: MB type, MB availability, MVD value (Table 5.1), predict direction of sub-MB of reference blocks (Table 5.3)	
Reference index	Reference sub-MBs: MB type, MB availability, slice type, prediction direction (Table 5.2), reference index (Table 5.2)	
CBF	DC block	Reference MBs: MB type, MB availability, CBF of DC block
	Other category	Reference blocks: MB type, MB availability, CBF of reference block (Table 5.1 for CBP luma, Table 5.2 for CBP chroma)

Additional required parameters of each neighboring reference BPMB are listed in Table 5.5, including MB type and MB availability of reference BPMB. If reference BMPM is located in neighboring MB, MB availability is derived based on the position of current MB. Top reference MB is not available if current MB is in the first row of coding frame, while left reference MB is not available if current MB is in the first column.

In IC sub-unit, the 3-bit *CtxIdxInc* is calculated based on SE values and other parameters obtained from reference BPMBs and parameters of current SE parsed from input packet. The calculation of each type of SE is implemented as combinational circuit. For the calculation that needs to access coded SEs of top and left reference BPMBs, SE access of two BPMBs are executed in parallel to reduce critical path length. For the calculation of non-residual SEs, *CtxIdxInc* is calculated and written to FIFO2 in the same cycle when the input packet is parsed. For CBP processing of non-P8×8 MB (4:2:0 video format), 6 *CtxIdxInc* values are generated from single input SE packet in 6 consequential cycles including 4 for CBP luma and 2 for CBP chroma. For MB of P8×8 mode, CBP bit of each 8×8 sub-MB is input separately with sub-MB index concatenated in the same packet, and one *CtxIdxInc* is generated. For the calculation of residual SEs, only CBF of each 4×4 block is processed in unit CS_1. According to the block index and 3-bit residual block category

of EOB (end of block: indicating the last packet of one residual block) packet of residual block, *CtxIdxInc* of CBF is calculated. The critical path of IC sub-unit is in the combinational circuit of *CtxIdxInc* calculation of MVD, which consists of two sequential table lookup operations, one 6-bit addition, and 7-bit comparison, and several other bitwise operations. Calculation of MVD is more complex as the context model is selected based on the evaluation of sum of two absolute MVD values of neighboring BPMBs.

5.1.2.2 Storage of Coded SE Values During CtxIdxInc Calculation

During *CtxIdnInc* calculation of IC sub-unit, SE value parsed from input packet needs to be stored in local buffer and SE RAM if the value can be further referenced during *CtxIdxInc* calculation of current MB or following MBs in the same slice. SE type, number and position of SEs, and number of bits per SE of each type of SE to be stored in the top and left reference MBs are discussed previously in Table 5.4. For several types of SEs with large data range of SE values including MB type, MVD, reference index, etc., original SE values are classified and remapped to smaller data set to reduce storage size of code SEs in local buffers and SE RAM. Storage operations of several types of SEs including MB type, MVD, CBF, etc. are discussed briefly in this section.

Values of MB type are classified into 7 categories as shown in Table 5.6, including 3 categories for intra MB (Intra4×4, Intra16×16, and I_PCM), 2 categories for inter MB (P8×8 and Skip), 1 for B slice (Direct), and 1 for other types of inter MB. During parsing and processing of MB type and sub-MB type, 2-bit prediction direction (forward, backward, and bi-direction) of each 8×8 sub-MB of inter MB is stored. For MB type of Intra16×16 in I, P, and B slice, 6-bit CBP value is derived from MB type value through table lookup of a small 6-entry LUT, while for other types of MB, CBP is parsed and stored from packets of CBP or CBP bit.

Table 5.6 Classification of MB type and stored values of MB type

MB type	3-bit Stored value	Descriptions
Intra4×4 (I4)	0	
Intra16×16 (I16)	1	
I_PCM	2	Intra-coded MB in I, P, and B slices
P8×8	3	In P and B slice, when MB partition mode of 8×8 or smaller partition is selected
Skip	4	In P and B slice, based on coding value of MB skip flag before MB type coding
Direct	5	MB type only in B slice coding
Other types	6	Inter-coded MB in P and B slice, excluding P8×8

Table 5.7 Numbers and positions of blocks that need to store coded MVD of different MB/sub-MV partition modes

MB partition mode	Num of blocks stored	Block index	Sub-MB partition mode	Num of blocks stored	Block index
16×16	7 of 16	5,7,10,11,13,14,15	8×8	3 of 4	All exclude top left block
16×8 Top	3 of 8	2,5,7	8×4	2 of 2	All
16×8 Bottom	5 of 8	10,11,13,14,15	4×8	2 of 2	All
8×16 Left	3 of 8	1,10,11	4×4	1 of 1	All
8×16 Right	5 of 8	5,7,13,14,15			

For the SEs of block/sub-MB level, only the SEs of blocks/sub-MBs located on the right edge and bottom edge of current BPMB need to be stored, because SE of other block/sub-MB can not be referenced during processing of following BPMBs of current MB or following MBs. As shown in Table 5.7, for MB partition mode including 16×16, 16×8, and 8×16, and sub-MB partition mode 8×8, MVD vectors (forward/backward, x/y) of current BPMB are only stored in part of blocks. Dynamic power consumption of MVD processing can be saved when power reduction technique is applied to unit CS_1.

Storage of CBF of each residual block is triggered by the parsing of EOB packet, and CBF value is located from buffer based on 4-bit block index and 3-bit block category, which classifies block of luma DC coefficients, luma AC coefficients, luma coefficients of non-Intra16×16 MB, chroma DC coefficients, and chroma AC coefficients. In the local buffer of IC sub-unit, 49 bits are allocated to buffer CBF values of current MB and 2 reference MBs, including 9 bits for CBF of DC blocks (3 for current MB), 24 bits for luma blocks of non-Intra16×16 MB (16 for current MB); and 16 bits for chroma AC blocks (8 for current MB).

5.1.2.3 End-of-MB Process

The last input packet of each MB is defined as MB end flag for the proposed encoder. End-of-MB process of IC sub-unit is timing efficient. When the MB end flag is parsed in unit CS_1, the design supports that the input packet of next MB to be received and processed in the next cycle. Two operations of End-of-MB process are triggered in IC sub-unit simultaneously, including (a) loading of coded SEs of next top reference MB from SE RAM output, and (b) loading of coded SEs of next left reference MB from local buffer of current MB. The coded SE buffers of top and left MB are loaded for the next MB in single cycle, and operation delay of coded SE loading is minimized compared to the reported CABAC decoder design [Chen05]. During processing of current MB in [Chen05], it takes 96 cycles to load SEs of top reference MB from RAM for the next coding MB and write coded SEs of previous

MB to RAM. Limitation of such scheme is that, RAM access frequency and related power consumption is high, and for the situation that the coding MB only contains a few packets, such as Skip MB, CABAC encoding stalls until data transfer between RAM of coded SE and context model selection module completes. Compared to [Chen05], the situation is avoided in the proposed encoder design.

5.1.3 The Memory Access (MA) Sub-unit of Unit CS_1

Coded SEs of top reference MBs are stored in SE RAM of Block 1. 141 bits are allocated in SE RAM to store coded SEs for each MB that can be referenced during MB coding. All 141 bits of one MB are stored as one word of RAM to enable single cycle read/write RAM access. Word number of SE RAM is proportional to the horizontal definition of coding picture, as coded MBs from top-right MB of current coding MB will be referenced as top reference MBs in further coding procedure. In HDTV 720p video format, each row of MBs of picture contains 80 MBs. In order to support HDTV 720p, 78 words are allocated in SE RAM with another 141-bit output buffer at RAM interface. It is designed as single-port RAM supporting sequential read and write access, as circuit area is small and SE RAM is not frequently accessed in the proposed coded SE access scheme.

MB processing flow of MA sub-unit and IC sub-unit are shown in Fig. 5.5. MA sub-unit is only active in the first 3 cycles of processing procedure of one MB. The process of MB_i is shown in the figure. In Cycle 1, after parsing MB end flag of MB_{i-1}, MA sub-unit requests to read coded SEs of top reference MB of MB_{i+1} from SE RAM, while IC sub-unit is triggered for the End-of-MB process including update of SE values of top and left reference MBs of MB_i. In cycle 2, MA sub-unit writes a single packet of 141 bits of coded SEs of MB_{i-1} to SE RAM, while IC sub-unit starts MB processing procedure including input packet parsing, *CtxIdxInc* calculation, and storage of coded SEs of MB_i. In cycle 3, MA sub-unit updates RAM address for the read of MB_{i+2} and write of MB_i. After the first 3 cycles, MA

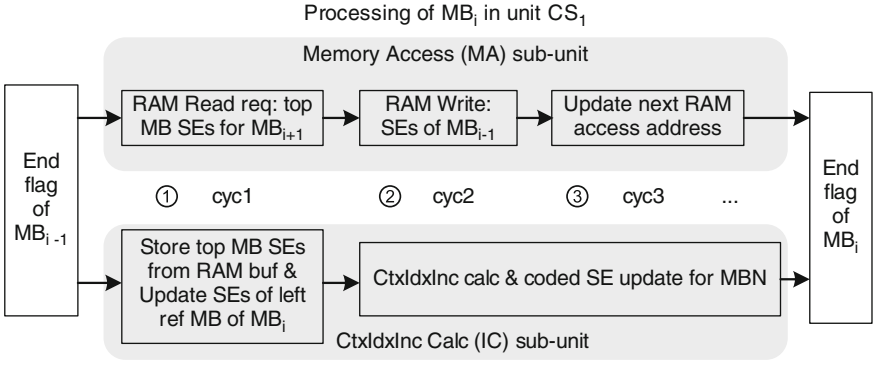

Fig. 5.5 MB processing in MA sub-unit and IC sub-unit of unit CS_1

sub-unit is idle, while IC sub-unit continues processing following packets of MB_i till the end flag of MB_i.

One example of 3-cycle MB processing scheme of MA sub-unit is shown in Fig. 5.6. N is MB row number. M is number of MBs in the horizontal direction of coding picture (M is 80 in HDTV 720p video format). $MB_{i,j}$ denotes MB with index j in row i. MBs of row N are processed sequentially. In cycle 1, the end flag of $MB_{N,M-2}$ is parsed in unit CS_1. At the end of $MB_{N,M-2}$ processing, MBs from $MB_{N,0}$ to $MB_{N,M-3}$ are stored in M-2 words of SE RAM, MB $_{N-1,M-1}$ is already read to the output buffer of SE RAM, and SE RAM is addressed at the word of $MB_{N,0}$. The operations of MA sub-unit in cycle 1 include: storing $MB_{N-1,M-1}$ of RAM output buffer to the top reference MB of $MB_{N,M-1}$ and reading the word of MB $_{N,0}$ to RAM output buffer. In cycle 2, the word of $MB_{N,0}$ is stored to RAM output buffer, which will be referenced during $MB_{N+1,0}$ processing; and MA sub-unit writes the coded SEs of $MB_{N,M-2}$ to the RAM and overwrites the word of

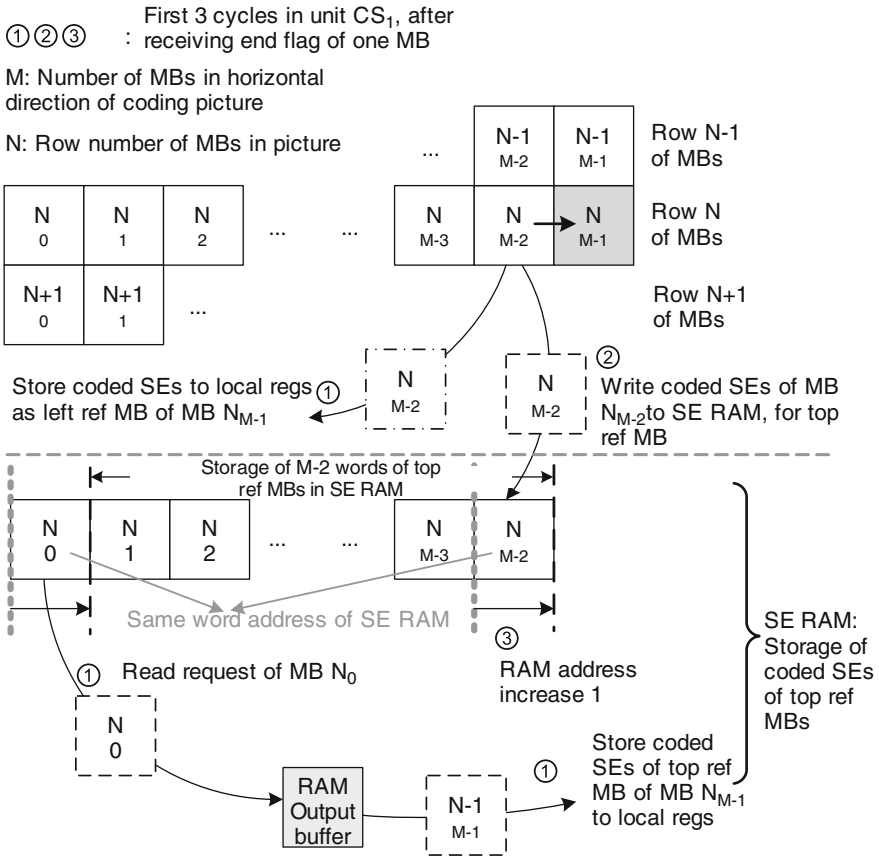

Fig. 5.6 Operations of MA sub-unit in the first 3 cycles of $MB_{N,M-1}$ processing

$MB_{N,0}$. In cycle 3, SE RAM address increases 1 in MA sub-unit, which points to the word of $MB_{N,1}$ that will be accessed after $MB_{N,M-1}$ processing.

The benefits of SE RAM access scheme of MA sub-unit includes: (a) RAM access frequency and related dynamic power consumption of SE RAM is minimized compared to the other designs [Chen05, Liu07a] that support context model selection of unit CS_1, because the RAM is read and write only once per MB; (b) compared to [Chen05], RAM size is reduced with the proposed RAM read and write access order, and only M-1 words of reference MBs need to be stored in SE RAM including that of output buffer; (c) allocation of RAM output buffer and schedule of reading next top reference MB during processing of current MB insure that no delay of coded SE access will occur between processing MBs, and throughput of CABAC encoding is not influenced by the access of coded SE, compared to [Liu07a]; (d) RAM access control is simplified, as addresses of RAM read and write operations are identical in cycle 1 and cycle 2 of MA sub-unit.

5.2 Unit CA: Efficient Context Model Access

For each regular bin of CABAC encoder, one context model must be read from context RAM to provide MPS value and probability index of LPS of the bin. Each accessed context model is updated according to bin value and written back to context RAM in order to be adaptive to the change of MPS/LPS probability. Scheme of single context model access is adopted in most reference designs [Ha05, Shojania05, Li06a, Nunez06, Chen07b, Liu07a], in which one context model (7 bits) is fetched from RAM each time according to *CtxIdx* of regular bin. Although memory control logic of single context model access is simple, RAM access frequency and related dynamic power are high, which significantly increases power consumption of CABAC encoder, as discussed in [Kuo06]. Moreover, control logic of context model access is complex, because parsing of multiple 9-bit RAM addresses is required to decide whether context model should be read from RAM or local buffer, and critical path of CA can not be reduced efficiently.

5.2.1 Context Line Access and Local Buffering

In this book, an efficient context model access scheme is proposed and implemented in unit CA, as shown in Fig. 5.7. Before design of unit CA, computational and data transfer complexity analyses have been performed on a reference CABAC code [Ho06], and it is found that memory accesses in CABAC entropy coder (as opposed to CAVLC), increases by 20–44% for RDO-off mode, and a higher of 104–115% for a RDO-on mode. Therefore, the use of multi-level memory hierarchy is suggested. Instead of using single context model access scheme adopted in [Ha05, Shojania05, Li06a, Nunez06, Chen07b, Liu07a], the proposed CABAC encoder accesses the context models of context RAM by context lines, each of which contains eight 7-bit

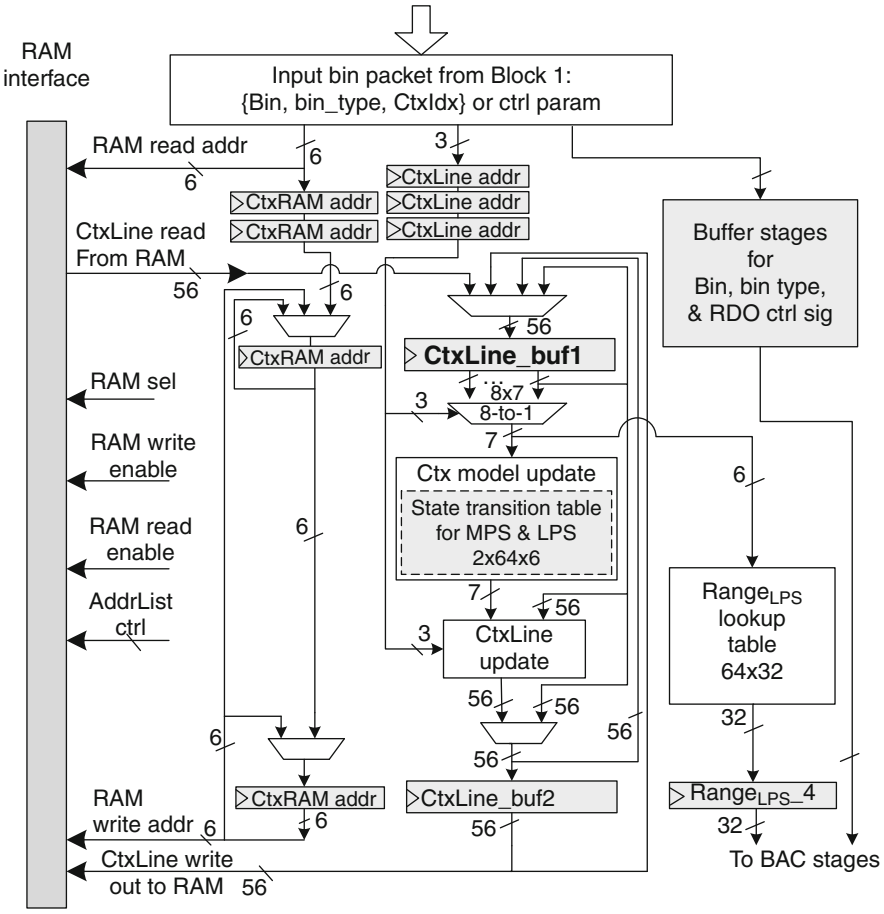

Fig. 5.7 Architecture of unit CA with pipelined context line access and local buffering scheme

context models. Two context line buffers are allocated in unit CA to buffer the context models read from context RAM. The difference from the cache-based context access in [Kuo06, Osorio06] is that, any context line, which contains the required context model, is pre-fetched from RAM before the context model is used. The situation of cache miss will not occur in this design.

In Fig. 5.7, the 9-bit *CtxIdx* of input bin packet of regular bin is separated into context RAM address (higher 6 bits of *CtxIdx*), and context line address (lower 3 bits of *CtxIdx*) in unit CA. Context line is accessed from RAM and updated in a 4-stage pipeline. Context RAM address parsed from input packet is used to locate one context line in the context RAM. It is buffered in 4 pipeline stages in unit CA for RAM read and write access. Before one context line is read from context RAM, the corresponding context RAM address is buffered in 2 cycles (2 stages). Context line address parsed from input packet is used to locate one of 8 context models in

the context line. It is buffered in 3 pipeline stages before it is used. As shown in the figure, two context line buffers CtxLine_buf1 and CtxLine_buf2 of unit CA can buffer 16 different context models during CABAC encoding. The 6-bit pStateIdx of selected context model from CtxLine_buf1 is used to prefetch 4 possible sub-Range values of LPS bin ($Range_{LPS}$, 8 bits) for unit AR through a 64-word×32-bit LUT. The pStateIdx of context model is updated through table lookup based on whether the bin is MPS or LPS. The two 64-word×6-bit LUTs of MPS bin and LPS bin are implemented as combinational circuits. Coding bin, bin type, and RDO control parameters parsed in unit CA are buffered in several stages and output to unit AR and unit BP. For regular bin, 1-bit decision of whether the bin is MPS is output to unit AR instead of the value of bin.

RAM access frequency of the proposed scheme is significantly lower compared to that of single context model access scheme. A context line needs to be read from context RAM and buffered in CtxLine_buf1 only when the accessed context model is not located in the two context line buffers, in which situation the content of CtxLine_buf2 is written back to context RAM. Since the context models of the same SE type are located in proximity in the context RAM, and in many situations, context models consecutively accessed are located in the same or neighboring context line, the frequencies of RAM read and write access can be significantly reduced with the context line access and buffering scheme, especially during the processing of residual SE such as SCF, LSCF, and abs_level.

5.2.2 Context RAM Access Scheme Supporting RDO-on Mode

In RDO-on coding mode, the state of context models in the context RAM needs to be restored to the previous state after testing one RDO mode. One scheme to support RDO-on coding is to store context line and the corresponding context RAM address into a FIFO buffer before the context line is updated during RDO coding, and restore the updated context lines of context RAM after RDO coding according to the stored context RAM addresses of the context lines. However, in non-P8×8 RDO coding, large number of context lines can be accessed. Thus, large FIFO buffers are required to backup the original state of context lines, and operation delay is not avoidable to restore of state of context RAM from FIFO buffer. Because context RAM is accessed by context line, delay of context state restoration can be reduced compared to the scheme in [Nunez06]. In this book, a more efficient context RAM access scheme is adopted in the design of unit CA with no operation delay in non-P8×8 RDO coding, as shown in Fig. 5.8.

In RDO-off mode, context models are stored in and accessed from Normal RAM. During RDO-on coding, Temp RAM, which is identical to Normal RAM, stores the updated context lines while Normal RAM keeps the original context lines. During coding of each RDO mode, if the context line is not accessed previously, it is read from Normal RAM, while updated context lines are stored to and accessed from Temp RAM. During non-P8×8 RDO coding, because Normal RAM is unchanged, context state restoration operation is not needed, and the operation delay is removed. In order to identify which context lines have been accessed during coding of one

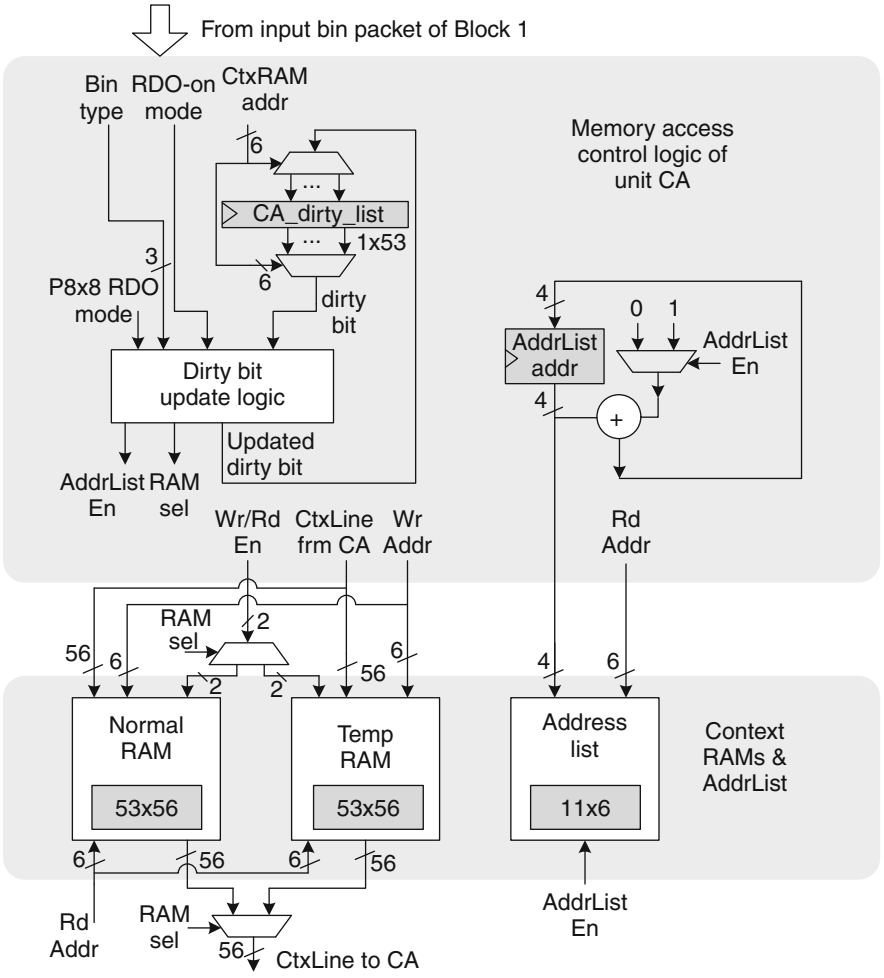

Fig. 5.8 Architecture of memory access control of unit CA in both RDO-off and RDO-on mode

RDO mode, a 53-bit dirty list is allocated in unit CA and controlled by a dirty bit update logic. Each bit of the dirty list records whether the corresponding context line is updated. One context line is read from Normal RAM only if the dirty bit of the context line is 0.

In non-P8×8 RDO coding, because the updated context lines of Temp RAM will not be further used after coding of current RDO mode, it is not needed to write the two context lines that are buffered in unit CA to Temp RAM when the context line access of current RDO mode completes. Therefore, context RAM write frequency can be reduced in non-P8×8 RDO coding for the proposed context line access scheme.

During P8×8 sub-MB mode decision of RDO-on mode coding, the updated context lines of one RDO mode need to be stored, if the mode is currently the best mode of sub-MB. Therefore, a small 11-word×6-bit RAM named Address list is allocated to store the context RAM address of each updated context line. Address list is accessed by Block 3 during operations of context state backup and restoration of P8×8 RDO coding.

5.2.3 Context Model Reallocation in Context RAM

In order to further reduce memory access frequency, context models in the context RAM are reallocated, as illustrated in Fig. 5.9. In the cache-based designs [Kuo06], context RAMs are also reallocated to enhance cache hit probability. In the proposed scheme, reallocation aims at enhancing the efficiency of context-line-based context model access. The context model allocations of different SE types are adjusted in the context RAM, compared to the original allocation in Table 9.11 of the standard [ISO/IEC 14496-10]. The objective is to allocate the context models of the same SE type in one context line to avoid unnecessary RAM access. For the SE type with over 8 context models, such as SCF, LSCF, and level, 2 context lines are allocated.

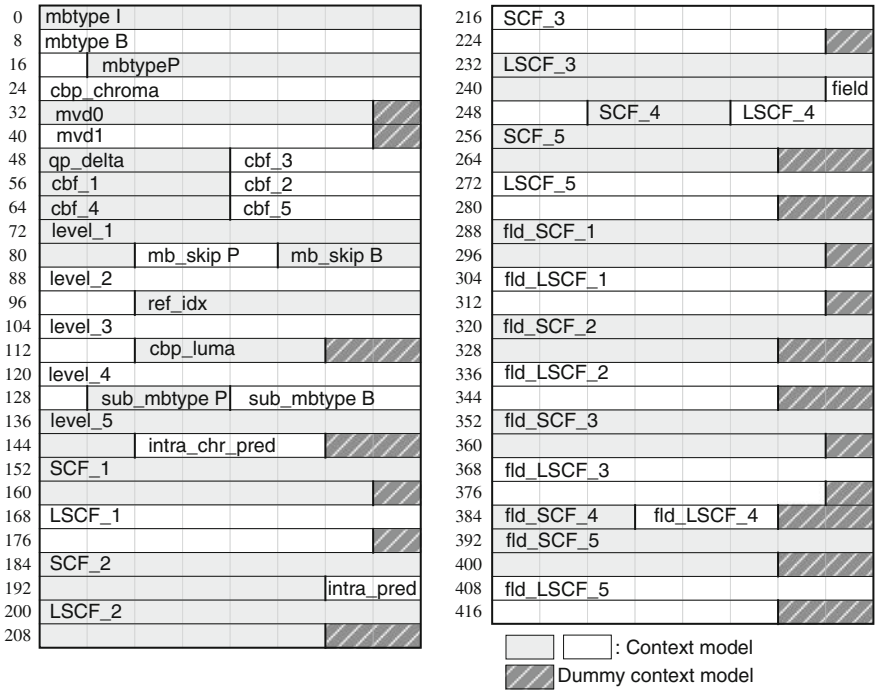

Fig. 5.9 Reallocation of context model in context RAM (Normal RAM). Context models of Normal RAM are illustrated as two continuous parts in the figure

53 context lines (424 context models) are allocated in the context RAM with only 6.8% dummy context models inserted.

5.3 Context State Backup and Restoration in P8×8 RDO Coding

As introduced in 0, during CABAC coding of RDO-on mode, different MB coding modes are tested, including Skip, intra modes including Intra16×16 and Intra4×4, inter modes including Direct, P16×16, P16×8, P8×16, and P8×8, etc. The best mode is selected that achieves lowest rate-distortion cost. During RDO MB mode selection, CABAC encoder is adopted to calculate coding rate (length of output bit stream) of each testing mode by encoding the SEs of the mode. Context state (state of context models in context RAM) needs to be restored to a previous state before testing of the next RDO mode. During RDO-on coding, large number of context models need to be frequently backed up or restored when RDO modes change. It causes long delay, and requires large amount of backup memory resources. With the aforementioned context access scheme of unit CA, the updated context models are stored in Temp RAM and context state of Normal RAM is not changed during non-P8×8 RDO coding. Therefore, it is not needed to backup and restore context state during non-P8×8 RDO coding, and the no operation delay is removed, compared to [Nunez06].

However, during P8×8 RDO coding, the best partition mode of one 8×8 sub-MB is selected from the modes of 8×8, 8×4, 4×8, and 4×4. During RDO mode decision of one sub-MB, context state needs to be restored to the state before testing of current sub-MB, after testing of each sub-MB partition mode. After testing all modes of the sub-MB, context state needs to be set to the state of best mode, which is the original state of next sub-MB of current MB. After processing of all 4 sub-MBs of P8×8 RDO coding, context state needs to be restored to the original state of current MB (the context state after RDO-off coding of previous MB). Because of the context state coherence of adjacent 8×8 sub-MBs during P8×8 RDO coding, it is indispensable to implement mechanisms of context state backup and restoration, compared to non-P8×8 RDO coding. In this design, it is found that a maximum of 11 context lines can be modified during P8×8 RDO coding of the Main profile of H.264/AVC. Three small RAM blocks of depth of 11 including Address list (11-word×6-bit), Best RAM (11-word×62-bit), and P8×8 RAM (11-word×62-bit) are utilized to store intermediate context states, including (a) addresses of the context lines accessed in each RDO mode stored in the Address list, (b) addresses and context lines of best P8×8 coding mode stored in the Best RAM used to update Normal RAM, and (c) addresses and context lines of original context state stored in the P8×8 RAM used to restore Normal RAM to the original state after P8×8 RDO coding. The CA dirty list in unit CA and P8×8 RDO dirty list in Block 3 are updated and referenced during RDO-on coding and RDO operations of context state backup and restoration.

As shown in Fig. 5.10, four pipeline architectures are designed in Block 3 to support 4 types of context state backup and restoration operations during P8×8 RDO coding with data transfer speed of 1 context line/cycle. Buffering stages of

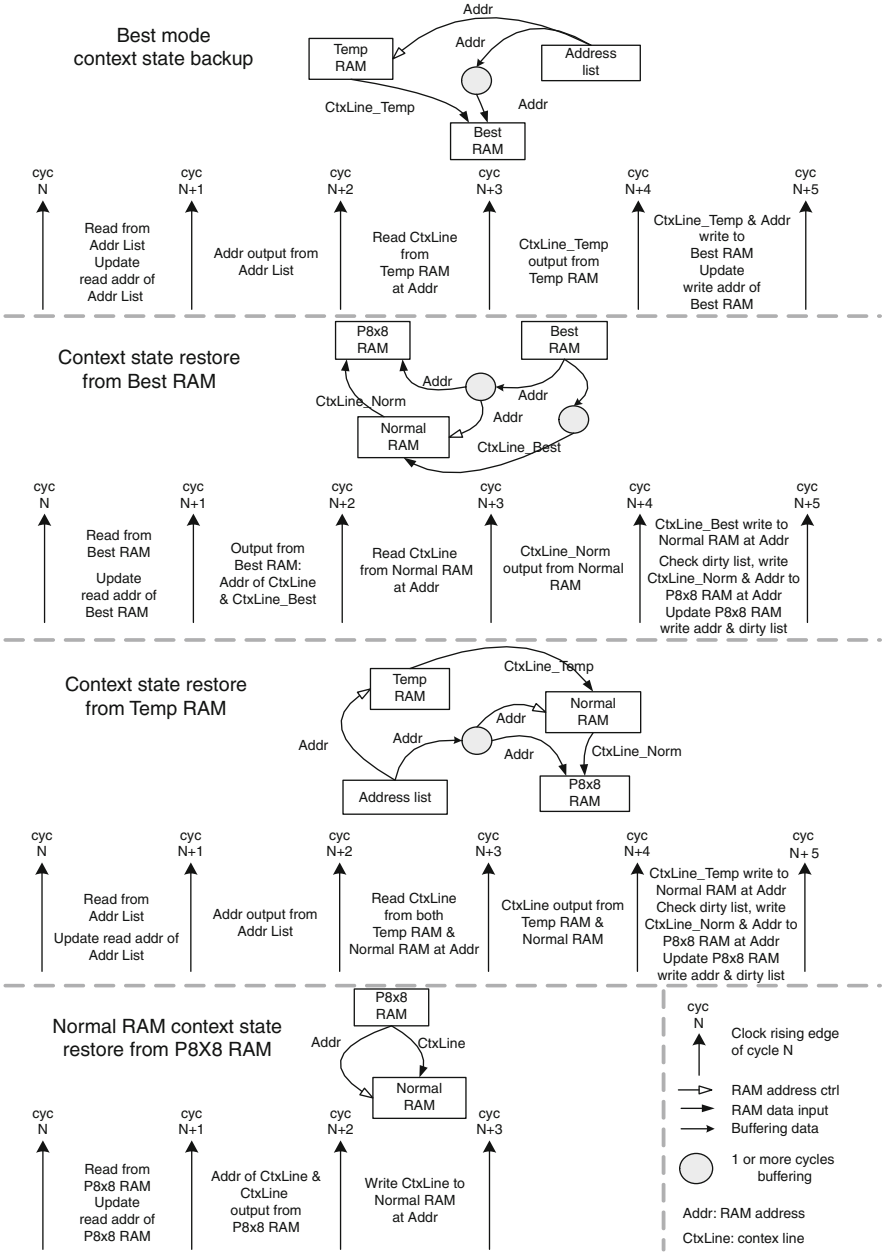

Fig. 5.10 Four types of pipelined context state backup and restoration operation in P8×8 RDO coding

context line and RAM address are inserted to implement pipeline operations. In the 2 context RAMs and 3 small backup RAMs, 3 to 4 RAMs are involved in each pipelined operation. Detail operations of each pipeline stage of the 4 pipelines are shown in the figure and introduced as follows:

- *Best mode context state backup*: If the RDO mode is currently the best mode of the processed 8×8 sub-MB, the modified context lines in the Temp RAM are fetched according to the stored RAM addresses in the Address list. The modified context line and corresponding RAM address are packed in single packet and backed up in the Best RAM;
- *Context state restore from Best RAM*: After testing all RDO modes of one 8×8 sub-MB, if the context state of the best mode is stored in the Best RAM, Normal RAM is updated by the context lines of Best RAM according to the RAM addresses stored with the context lines. The original context lines of Normal RAM are stored to P8×8 RAM with corresponding context RAM addresses before Normal RAM is updated by the context lines of Best RAM. Write access of P8×8 RAM is according to the state of 53-bit dirty list of Block 3.
- *Context state restore from Temp RAM*: If the best mode of current sub-MB is the last RDO mode, the updated context state is not stored in the Best RAM, but in the Temp RAM. Therefore, Normal RAM is updated by the context lines of Temp RAM that are accessed in the last mode according to the addresses stored in the Address list. Before the update, original context lines of Normal RAM are stored in P8×8 RAM with context RAM address if the corresponding bit of dirty list is 0.
- *Normal RAM context state restore from P8×8 RAM:* After P8×8 RDO coding, the context state of Normal RAM is restored to the original state by writing back the unmodified context lines stored in the P8×8 RAM according to the addresses that are packed with the corresponding context lines.

With the proposed pipelined context state backup and restoration and context line access scheme of context RAM, the operational delay of context state backup and restoration of P8×8 RDO coding is significantly reduced, and the context RAM size to support is P8×8 RDO coding is reduced to a large extent, compared to the reference design. Performance comparison of the proposed design and reference design will be discussed in Chap. 7.

5.4 Coded SE State Backup and Restoration of Unit CS_1

In RDO-on mode coding, state of coded SEs of one RDO mode need to be backed up if the SEs can be referenced during testing of following RDO modes of the MB. A particular situation is P8×8 sub-MB mode decision. Coded SEs of the best mode (selected mode) of current sub-MB can be referenced during RDO coding of block/sub-MB level SEs of following sub-MBs (on the right or bottom of current sub-MB) of current MB. Compared to the context state backup and restoration of

P8×8 RDO coding, it is only necessary to backup part of coded SEs of one sub-MB before the best mode is decided, and restoration of coded SE state is not required after processing of 4 sub-MBs. Table 5.8 illustrates the SE type and number of bits of coded SEs that need to be backed up during P8×8 mode decision of one sub-MB. For block level SEs including MVD and CBF, only coded SEs of right and bottom blocks of sub-MB need to be backed up, because SEs of top left block can not be referenced during *CtxIdxInc* calculation of following sub-MBs. Coded sub-MB level SEs including prediction direction and reference index are backed up for the coding sub-MB. For the MB level SEs of CBP, single bit is necessary to record the CBP bit of best mode of current sub-MB, and one of 4 bits of CBP is updated after mode decision of the corresponding luma sub-MB.

Four types of coded SE state backup and restoration operations are supported, and can be triggered by the same control parameters that also control operations of context state backup and restoration during P8×8 RDO coding. (a) After coding of one RDO mode, if it is currently best mode (with minimum RD cost) of the processing sub-MB, coded SEs are backed up, including CBP bit, prediction direction, reference index, CBF and MVDs of 3 blocks of the sub-MB. (b) After all RDO modes of the sub-MB are tested, if the last mode is not best mode and the sub-MB is not the last of the MB, the coded SEs of best mode are restored, which were backed up in (a). (c) After all RDO modes of the sub-MB are tested, if the last mode is the best mode and the sub-MB is not the last of the MB, no restoration of coded SEs is needed. (d) After processing of all four 8×8 sub-MBs, values of CBP, CBF, prediction direction, and reference index are restored to the default values before coding of non-P8×8 RDO modes. During P8×8 RDO coding, if the RDO mode is not the best mode during current sub-MB coding, coded SEs including CBF, reference index, and prediction direction need to be restored to the default values.

During coding of other non-P8×8 RDO modes, some types of SEs need to be restored to initial values. For instance, CBF needs to be restored after testing each Intra4×4 RDO mode, and in non-P8×8 RDO inter mode, prediction direction values of sub-MBs need to be restored to default values after the RDO mode is tested. These types of coded SE restorations are triggered by the RDO coding rate output

Table 5.8 Types, bit numbers, and usage descriptions of backup values of SEs of 8×8 sub-MB during P8×8 RDO coding

SE type	Num of bits	Description of usage
MVD	3×4×7=84	MVDs for 3 blocks of sub-MB, excluding top left block, forward/backward directions, x/y components
CBF luma	3	CBFs for 3 blocks of current sub-MB, excluding top left block
CBP	1	Used in P8×8 RDO coding, CBP bit of current sub-MB of best mode
Pred dir	2	Prediction direction for current sub-MB
Ref index	2×1	Forward/backward direction, 1 bit each

signal of input packet, which is sent to unit BP of Block 2 after all SE packets of the RDO mode are processed.

5.5 Summary

In this chapter, the most complete context modeling scheme for CABAC encoder is proposed for the context model selection and context model access in both RDO-on and RDO-off modes. For context model selection, single-cycle *CtxIdxInc* calculation of regular bin is achieved for all types of SEs, which can be attributed to the efficiency of the proposed scheme of storage and fast access of coded SEs of current MB and reference MBs. In addition, RAM access frequency of the coded SEs is significantly reduced, compared to the reported designs [Chen05, Liu07a]. For context model access, a pipelined context line access and local buffering scheme is proposed, which can be significantly reduced context RAM access frequency. In order to fully support RDO in CABAC encoder, operations of backup and restoration of context state and coded SE state are implemented in the 3 functional blocks of CABAC encoder. Coded SE state backup and restoration is only supported in the proposed design, and context state backup and restoration is more efficient compared to [Nunez06], with significant lower operational delay and smaller sized context RAMs.

Chapter 6
Design of System Bus Interface and Inter-connection of SoC-Based CABAC Encoder

As introduced in Chap. 4, one step of the SoC-based entropy coder design flow is SoC features introduction. Design of system bus interface for the CABAC encoder enhances portability and reusability of the design in the SoC video encoder system. In this chapter, design of master and slave interfaces of WISHBONE [Wishbone] system bus of CABAC encoder is discussed after the introduction of WISHBONE system bus specifications. The crossbar system bus inter-connection is also designed, which enables convenient integration of the proposed CABAC encoder with other IP blocks in the H.264/AVC encoding system.

6.1 Introduction of the WISHBONE System Bus Specifications

WISHBONE system bus provides a flexible design methodology to create a common interface among the IP cores of System-on-Chip (SoC). By adopting the standard inter-connection scheme of WISHBONE and designing system bus interface of each IP core, the IP cores can be integrated more quickly and easily. The goal of WISHBONE bus is to provide robust inter-connection that is complete compatible among IP cores and is not constrained the creativity of core developers or end users.

6.1.1 Interface Signals of the WISHBONE System Bus

The simplest type of interconnection is point-to-point that directly connects one master to one slave, as illustrated in Fig. 6.1.

The input signal of the interface is named with System reset (*rst_i*), and clock signals (*clk_i*) are generated by the SysCon module of the WISHBONE bus and input to all masters and slaves that are connected to the bus. For the other signals of WISHBONE bus, output signals of master correspond to the input signals of slave, and vice versa. The signals of WISHBONE master interface are listed in Table 6.1, and functions of the interface signals are described accordingly.

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6_6,

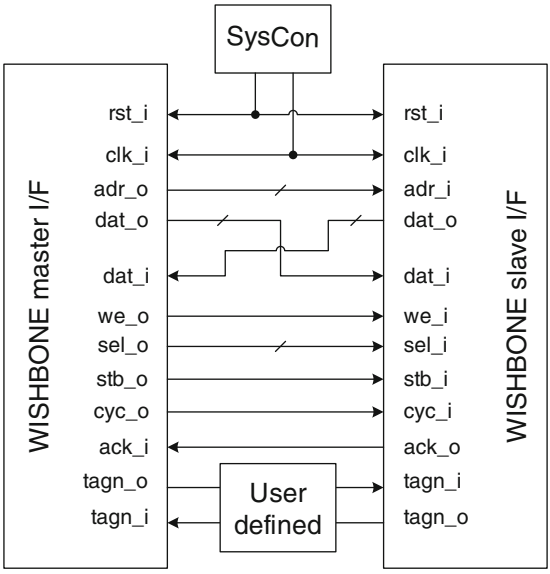

Fig. 6.1 Wishbone-based point-to-point inter-connection of single master and slave

Table 6.1 Signals of WISHBONE master interface

Signals of master interface	Function descriptions of signals of master interface	Corresponding signals of slave interface
rst_i	Interface reset signal	*rst_i*
clk_i	Clock of WISHBONE interface	*clk_i*
adr_o	The address of WISHBONE slave	*adr_i*
dat_o	Master output data bus	*dat_i*
dat_i	Master input data bus	*dat_o*
we_o	Write/read enable signal (high logic for write, low for read)	*we_i*
sel_o	Bit array that indicates valid data units on the data bus that master wants to send or receive data	*sel_i*
cyc_o	High logic indicates a valid bus cycle, which can contains multiple data transfers	*cyc_i*
stb_o	Strobe output that indicates a valid data transfer cycle	*stb_i*
ack_i	Cycle termination signal from slave for the acknowledgement of a successful data transfer	*ack_o*
rty_i and *err_i*	Cycle termination signal from slave indicating retry and error of the data transfer; the signals help to enhance robustness of data communication	*rty_o* and *err_o*
tagn_i and *tagn_o*	Tag values that provide additional information of data bus, address bus, and bus cycles	*tagn_o* and *tagn_i*

6.1.2 Types of Bus Cycles on the WISHBONE System Bus

Data transfer on WISHBONE is implemented using two types of bus cycles: (a) the classic bus cycles, and (b) the registered feedback bus cycles. A bus cycle is defined [Wishbone] as the procedure whereby digital signals affect the transfer of data across a bus by means of an interlocked sequence of control signals, and is differentiated from the clock cycle of signal *clk_i*. Each bus cycle contains one or several data transfer phases.

For the registered feedback bus cycles, signals of *ack_i*, *rty_i*, and *err_i* are registered at the slave interface, which cost one additional wait cycle during data transfer. However, the delay of feedback loop is significantly reduced and the cycles are suitable for high speed applications. In addition, the WISHBONE interface that supports registered feedback cycles also supports classic cycles, and can communicate to the interface that only supports classic cycles.

6.1.3 Comparison of WISHBONE and AMBA System Buses

Compared to the de-facto standard on SoC system bus – AMBA (Advanced Microcontroller Bus Architecture) of ARM, WISHBONE has the following differences:

- Data bus configuration of WISHBONE is more flexible. Granularity is used to define the minimum unit of data transfer supported by the bus interface.
- Tagging technique (user defined tag) of WISHBONE provides methodology to modify or extend the function of system bus interface. Inter-connection configuration of WISHBONE IP cores is also more flexible, with multiple choices of inter-connection modes.
- The complexity of WISHBONE system bus is lower, and the circuit area is relatively smaller.
- WISHBONE inter-connection is loyalty free. In addition, many WISHBONE-compatible free IPs have been developed and can be adopted to implement particular functions in the same SoC environment.

Because of simplicity, flexibility, and loyalty free of WISHBONE, the proposed CABAC encoder is equipped with WISHBONE system bus interfaces to enable ease of integration with other HW IPs or host processor of H.264/AVC encoding system.

6.2 Design of WISHBONE System Bus Interfaces for CABAC Encoder

The functions of the proposed CABAC encoder can be summarized as: (i) parsing and encoding of SE values of input packets according to the control parameters parsed from input packets, and (ii) output packets of coded bit stream in RDO-off

mode coding and output coding rate of each RDO mode in RDO-on mode coding. Input packets of SE values and control parameters of CABAC encoding are generated by the host processor during MB encoding procedure, while the output packets of RDO coding rate need to be feedback to the host processor during RDO-on MB coding mode decision, and coded bit stream of CABAC encoder of RDO-off mode should be sent to the output buffer of H.264/AVC encoder. WISHBONE interfaces are designed to support packet input and output process of CABAC encoder. Functional partitioning and architecture of the interfaces are discussed as follows.

6.2.1 Functional Specifications of the WISHBONE System Bus Interfaces

Reception of input packets and outputting of coded bit-stream can be implemented in one or two WISHBONE interfaces. For single interface design, the size of the CABAC IP block can be reduced as the interface signals are shared by both input and output modules, and requires single channel of the system bus. However, in each clock cycle, either read or write operation can be performed at the bus interface. In case that unit BP needs to output a packet and unit BN need to input a packet, one of the two operations needs to be paused, and the throughput of CABAC encoding pipeline is degraded. Furthermore, the system bus is occupied unnecessarily long. The reason is that: for a single master interface design, the master cannot predict when the next input packet will be available, and needs to keep *cyc_o* and *stb_o* high for request to read next input packet; and for a single slave interface design, host processor or the bit stream buffer equipped with master interface cannot estimate when the CABAC encoder will output next coded packet, and has to occupy the bus to keep read request.

The better functional partitioning scheme adopted in this book is to input packet through a WISHBONE slave interface and output packet through a master interface. Data transfer is controlled by the functional block which generates packets. Packet input is controlled by the host processor, and WISHBONE read cycle is triggered only when a input packet is available at host; on the other hand, packet output is controlled by the CABAC encoder, and write cycle is triggered when a output packet is ready in unit BP. Maximum coding throughput can be achieved in this partitioning scheme, and system bus occupation is minimized.

6.2.2 Analysis of Support of WISHBONE Registered Feedback Cycles

Both master and slave interfaces are designed to support registered feedback cycles of WISHBONE system bus in order to achieve high data rate required by CABAC encoding. The types of registered feedback cycles are identified according to the

Table 6.2 Type of register feedback cycles of WISHBONE classified by cti_o

3-bit value of cti_o	Type of registered feedback cycle
'000'	Classic cycle
'001'	Constant address burst cycle
'010'	Incrementing burst cycle
'111'	End-of-burst

3-bit cycle tag *cti_o* (cycle type identifier) generated at master interface. As shown in Table 6.2, the four types of registered feedback cycles are classic cycle, constant address burst cycle, incrementing burst cycle, and end-of-burst.

To enable easy integration of WISHBONE classic and registered feedback IP cores, it is required [Wishbone] that the WISHBONE interface with registered feedback of termination signal (*ack_i*, etc.) must support classic cycle (*cti_i* is "000"). Because *ack_i* is asserted one clock cycle after assertion of *stb_o* in registered feedback cycle, it requires 2 clock cycles to complete each data transfer for classic cycle, and the maximum throughput is 1/2 data transfer per clock cycle.

Figure 6.2 illustrates the signals of master interface of one classic cycle in 5 clock cycles from cyc0 to cyc4, with two data transfers (write, and then read). Slave interface asserts *ack_i* in cyc1 to indicate that the master *dat_o* will be latched at clock edge of cyc2. Because there is no information about what master will do in cyc2, slave negates *ack_i* in cyc2. In cyc2, master requires to read data. In cyc3, slave asserts the registered *ack_i*, and prepares read data on *dat_i*. At clock edge of cyc4, master latches *dat_i* and negates *stb_o* and *cyc_o*, and the classic cycle completes.

Throughput of constant address burst cycle and incrementing burst cycle are significantly higher, compared to that of classic cycle. Assume burst length of the cycle is N, it takes N+1 clock cycles to complete N data transfers instead of 2N clock cycles of classic cycle. The throughput of burst increases and approximates 1 when the burst length N increases. Therefore, supporting of burst cycle is necessary for high speed data transfer. End-of-burst indicates that the next transfer is the last of current burst. It is required to support end-of-burst if any of the two types of burst cycles is supported.

For the proposed design of WISHBONE interfaces, because function of packet input and output of CABAC encoding are partitioned to the two interfaces, it is only necessary to support write cycle that receives input packets at slave interface and to support write cycle that sends out packets of coded bit stream and RDO rate at master interface. In CABAC encoding, packet output frequency is significantly lower than packet input frequency. The master interface is designed to support classic cycle, while the slave interface supports not only classic cycle, but also constant address burst and end-of-burst. As single address of slave interface is accessed, it is no need to support incrementing burst cycle, in which consecutive addresses are accessed sequentially during data transfer.

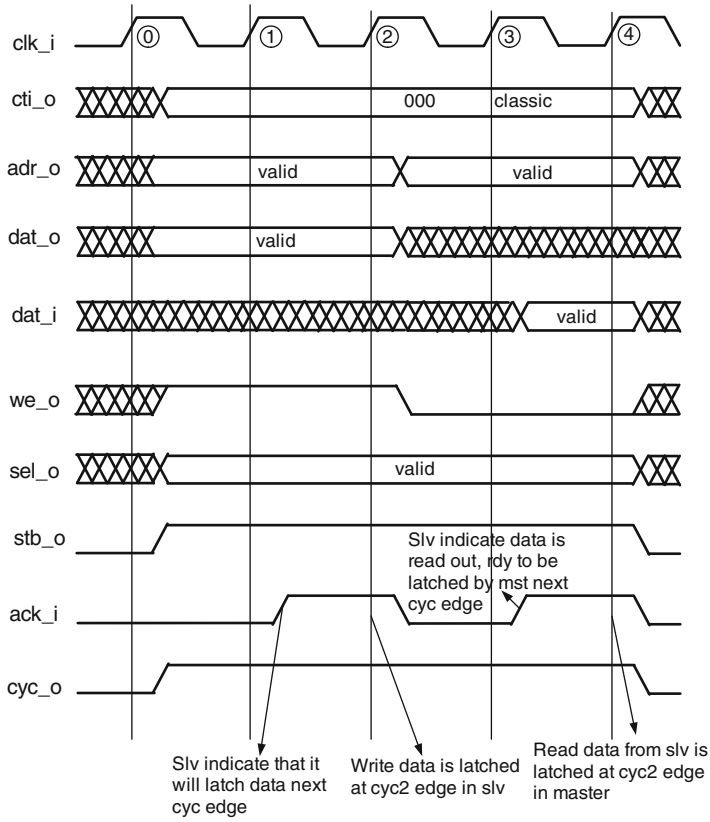

Fig. 6.2 One classic cycle of a WISHBONE master interface with registered feedback of cycle termination [Wishbone]

6.2.3 Design of Slave Interface of WISHBONE System Bus

The slave interface of WISHBONE system bus is designed to receive the input packets of CABAC encoder. It supports 3 types of registered feedback cycles including classic cycle, constant address burst cycle, and end-of-burst. The host processor equipped with WISHBONE master interface controls the generation and transfer of input packets of CABAC encoder. Input packets received by the slave interface are buffered in FIFO1 of CABAC encoder, which is accessed by functional units of binarization and context model selection. Insertion of FIFO1 enables input packing receiving of slave interface and following packet parsing and processing steps to execute in parallel, with higher average throughput. An example of burst bus cycle of WISHBONE slave interface of CABAC encoder is shown in Fig. 6.3.

The cycle involves 8 clock cycles from cyc0 to cyc7. The 3-bit value “001” of cti_i indicates that it is a constant burst cycle. Signal *ack_o* of slave is asserted in cyc1, which is one clock cycle after assertion of *stb_i*. Input packet on *dat_i* is latched at the next clock edge if *ack_i* is asserted. In this example, 4 input packets

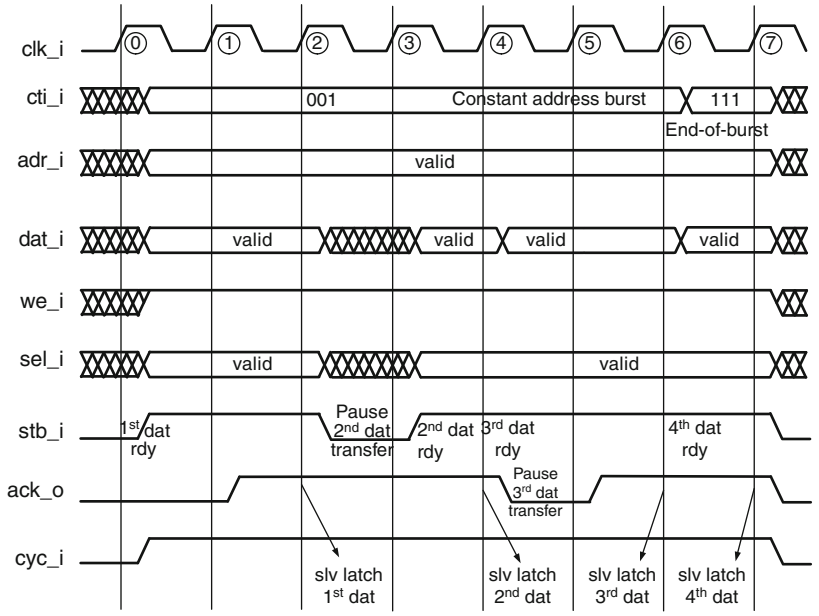

Fig. 6.3 Illustration of constant address burst cycle of WISHBONE slave interface [Wishbone]

are transferred from host processor and latched by the WISHBONE slave at clock edge of cyc2, cyc4, cyc6, and cyc7. In cyc6, *cti_i* is set to "111" indicating end-of-burst, and the burst completes in cyc7. It is also shown in the figure that both master and slave can suspend data transfer by negating *stb_i* or *ack_o*, respectively. Thus, packet input speed can be constrained by the master and slave interfaces. Assertion of *ack_o* of slave interface depends on the following conditions: (a) FIFO1 of input packets is NOT nearly full, so that the next input packet can be buffered; (b) the slave interface is addressed; and (c) *cyc_o*, *stb_o*, and *we_o* are of high value, indicating master requests a write operation.

For the interface signals, the width of input data bus *dat_i* is 22 bits, which is the size of input packet of CABAC encoder. The 4 MSBs of address bus *adr_i* are allocated to identify the slave devices connected to the WISHBONE system bus, and 16 slave devices can be identified. If the WISHBONE master and slave devices are connected through a system bus inter-connection block (INTERCON), the address bus between INTERCON and slave interface can be removed by applying partial address decoding scheme, in which *adr_i* are decoded in INTERCON.

6.2.4 Design of Master Interface of WISHBONE System Bus

Master interface of WISHBONE system bus is designed to output packets of coded bit stream or rate of each tested RDO mode, which are generated in the two pipeline stages of unit BP (bit packing) of CABAC encoder. As aforementioned, the rate

of output packets is significantly lower compared to that of input packets during CABAC encoding. The interface is designed to support only one type of registered feedback cycles: classic cycle. It is not efficient in timing and area to support burst cycle when burst length is only 1, because the timing efficiency is same compared to that of classic cycle, and interface complexity is higher to support burst. In CABAC encoding procedure, the probability is low to output packets of coded bit stream in continuous clock cycles. Therefore, burst cycle is not supported in the proposed system bus master interface of CABAC encoder.

Coded results of CABAC encoder are input to the master interface from unit BP, including 13-bit RDO rate in RDO-on coding and coded bytes in RDO-off coding. Trigger signals are generated in unit BP to notify the master interface whether the output is RDO rate or coded bytes, and during RDO-off coding, additional flag (*EOS_true*) indicates whether the output bytes are of EOS (end of slice) coding. It is shown in Table 6.3 the output order of coded bytes according to the values of *EOS_true* and Byte index. Excluding the confirmed OS (outstanding) bytes, 1–2 coded bytes can be output when *EOS_true* is 0, and 2–3 bytes can be output when *EOS_true* is 1, as the value of coding interval needs to be flushed out at the end of slice coding. The confirmed OS bytes are output after the first byte, and the number of OS bytes is based on a 3-bit counter in unit BP. In the WISHBONE master interface, coded bytes are packed on data bus *dat_o* and output in the order shown in Table 6.3.

Design of master interface with 32-bit data bus is shown in Fig. 6.4. The 32-bit width of data bus is commonly adopted in WISHBONE compatible IP design, which makes it easy to integrate with other IP cores. Four-bit *sel_o* is allocated to indicate the valid bytes on the bus during transfer. As shown in the figure, one of two processing paths is triggered during packet output in the master interface according to whether CABAC encoder is in RDO-off mode or RDO-on mode. Multiplexers are allocated to select the proper output signals of system bus from the two paths. During RDO-on mode, one data packet (*dat_o*) of RDO rate is output for each RDO mode; while in RDO-off mode, coded bytes can be output in 1 or more packets, including one packet of first byte, one to two packets of confirmed OS bytes, and one packet of remaining bytes. In the proposed packet output procedure, it is not needed to accumulate 4 bytes before output of 32-bit *dat_o*, as *sel_o* is utilized to indicate the positions of valid bytes of *dat_o*.

Table 6.3 Configuration of coded bytes output order of RDO-off coding

EOS_true	Byte index	EOS byte	Byte2	Byte1	Confirmed OS bytes (0–7 bytes)
0	0	N/A	N/A	1st byte	Output after Byte1
0	1	N/A	1st byte	2nd byte	Output after Byte2
1	0	N/A	1st byte	2nd byte	Output after Byte2
1	1	1st byte	2nd byte	3rd byte	Output after EOS byte

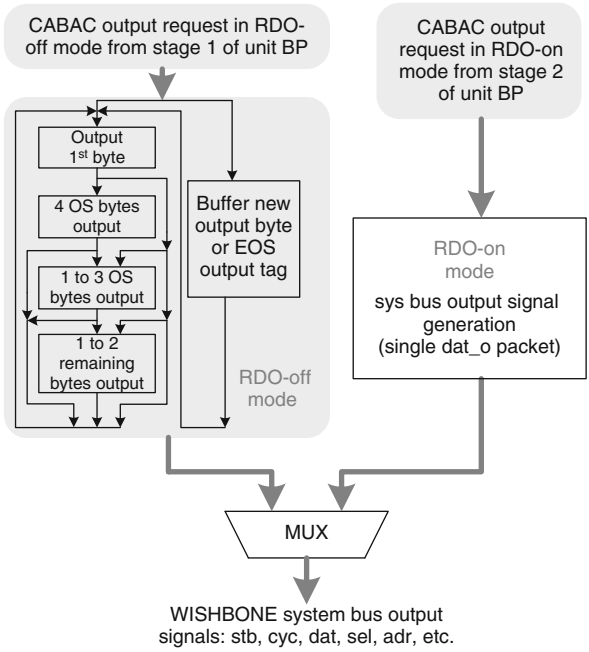

Fig. 6.4 Data output control of WISHBONE master interface with 32-bit *dat_o* bus

An alternative design of master interface is also implemented with *dat_o* bus width of 8 bits. As shown in Fig. 6.5, rate of each RDO mode is output in two sequential packets in RDO-on mode, and OS bytes are output sequentially in 1–7 packets in RDO-off mode. Control logic is simpler compared to that of master

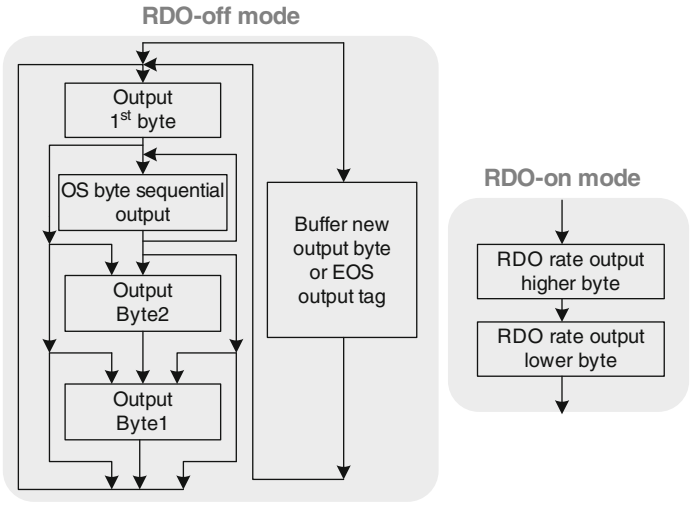

Fig. 6.5 Data output control of WISHBONE master interface with 8-bit *dat_o* bus

interface with 32-bit *dat_i*. However, coded byte output delay is longer, as maximum of 10 packets need to be output, and the integratability of the CABAC encoder equipped with the interface is lower.

6.3 Design of System Bus Inter-connection

System bus inter-connection is a key functional block of WISHBONE system bus that builds connection between masters and slaves according to a predefined manner. The block is named INTERCON in WISHBONE system bus specification [Wishbone]. Several inter-connection modes are supported in WISHBONE system bus. Point-to-point INTERCON only supports connection of a single master interface and a single slave interface, and is not suitable for SoC multi-device interconnection. Shared bus INTERCON supports connection of multiple masters and slaves. However, only single-channel connection is supported, and one master is allowed to initiate a bus cycle to a target slave through the connected channel in INTERCON. Data transfer rate of shared bus INTERCON is limited for high data rate video coding system. In comparison, crossbar INTERCON (crossbar switch) allows multi-channel inter-connection between masters and slaves. Each connection channel can be operated in parallel to other connection channels. This increases the data transfer rate of the entire system by employing parallelism. Crossbar INTERCON is inherently faster than traditional bus schemes. Crossbar routing mechanisms generally support dynamic configuration. This creates a configurable and reliable network system. The proposed crossbar INTERCON is introduced as follows.

6.3.1 Design of WISHBONE Crossbar INTERCON

The top-level architecture of WISHBONE crossbar INTERCON is shown in Fig. 6.6. The INTERCON supports connection with 4 masters and 4 slaves through system bus interfaces, and maximum of 4 connection channels can be established in the INTERCON if each master addresses a different slave. As shown in Fig. 6.6, the INTERCON consists of 4 sub-units (named M0, M1, M2, and M3) for the function of *master input selection and slave cyc generation* and 4 sub-units (named S0, S1, S2, and S3) for the function of *master connection arbitration and slave input selection*. Sub-units M0, M1, M2, and M3 can connect to the corresponding interfaces of 4 WISHBONE slaves through system bus, and sub-units S0, S1, S2, and S3 can connect to the corresponding interfaces of 4 WISHBONE masters. Each of M0–M3 generates cyc signals for the 4 slaves. Based on the 4 cyc signals from M0 to M3, each of S0–S3 sub-units makes arbitration on which master can connect to the slave that is connected to the sub-unit, generates Grant signals to M0–M3, and selects the WISHBONE input signals of the slave. Based on the Grant signals from S0 to S3, each of M0–M3 selects WISHBONE input signals for the master that is connected to the sub-unit.

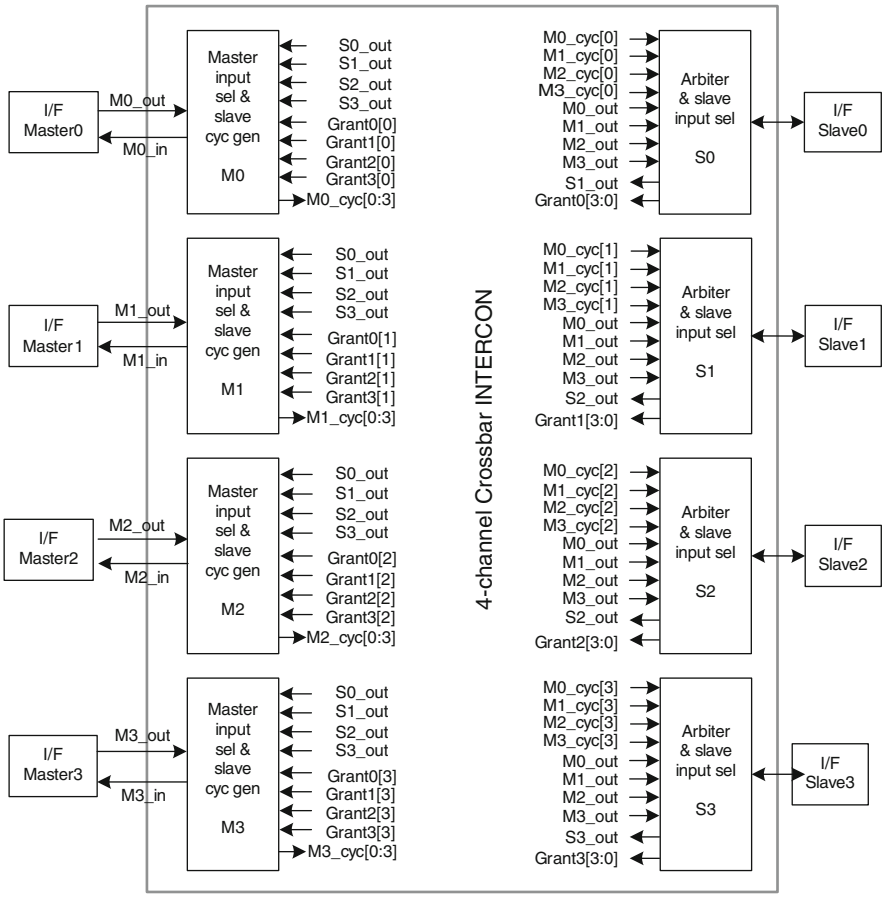

Fig. 6.6 Top-level architecture of 4-channel crossbar INTERCON of WISHBONE system bus

6.3.1.1 Master Connection Arbitration and Slave Input Selection

The architecture of *master connection arbitration and slave input selection* sub-units (S0–S3) of INTERCON is shown in Fig. 6.7. WISHBONE bus cycle signal cyc_o indicates the request of master to initiate a bus cycle to the addressed slave. For the same slave, bus cycle signals of 4 masters (M0_cyc, M1_cyc, M2_cyc, and M3_cyc) generated in M0–M3 are input to the arbiter of master connection. The bus connection of one of the 4 masters can be granted, and the 4-bit Grant signal is stored in registers. It is used to control system bus connection of the granted master to the slave in the multiplexer. The 4-bit Grant is also connected to M0–M3 for the selection of master input signals.

Round-robin arbiter and priority arbiter can be adopted for the arbitration of mater connection. Compared to round-robin arbiter, the limitation of priority arbiter is that, in very busy systems, there is no limit to how long a lower priority request

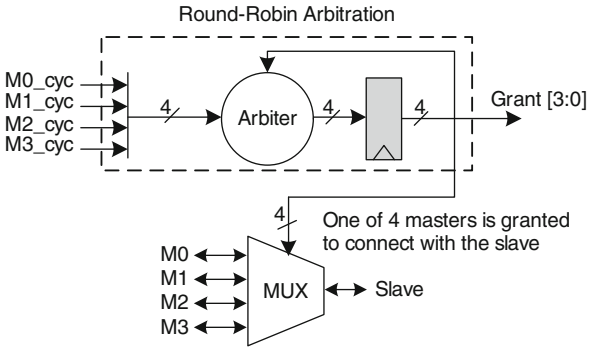

Fig. 6.7 Round-robin arbitration of system bus master that connects to the slave

may need to wait until it receives a grant [Weber02]. A round-robin arbiter on the other hand allows every requester to take a turn in order. The maximum amount of time that a requester will wait is limited by the number of requesters. In the proposed design, the arbiters in sub-units S0–S3 are designed as round-robin arbiters. If the currently granted master completes bus cycle and any of the remaining masters requests to initiate a bus cycle, the first requesting master in the round-robin order will be granted based on the stored Grant value.

6.3.1.2 Master Input Selection and Slave cyc Signal Generation

Architecture of master *input selection and slave cyc generation* sub-units (M0–M3) of INTERCON are illustrated in Fig. 6.8, with M0 sub-unit shown as an example. For the generation of bus cycle signal cyc of each of the 4 slaves that can connect to the master (Fig. 6.8a), address comparison of master address *adr_o* and the predefined address of the slave is carried out first. If the addresses are identical, and the master is requesting bus connection (*cyc_o* is high), the generated bus cycle signal M0_cyc of the slave is set high, which is used for master connection arbitration in S0–S3. As address comparison is completed in M0–M3, computation complexity of arbitration of S0–S3 is simplified. As shown in Fig. 6.8b, input data bus dat_i of each master is selected from *dat_o* of 4 slaves in multiplexer based on the Grant signals generated in S0–S3. Because a WISHBONE master can be granted to connect to only one WISHBONE slave, *ack_i* of the master is set high if any slave set *ack_o* high, and the master is granted to connect to the slave.

6.3.2 Compact SoC-Based CABAC Encoding System with WISHBONE System Bus Inter-connection

A compact SoC-based CABAC encoding system is shown in Fig. 6.9, which consists of the proposed CABAC encoder and WISHBONE INTERCON. Input packets of SE and control parameters are connected from interface of Master0 to M0 sub-unit

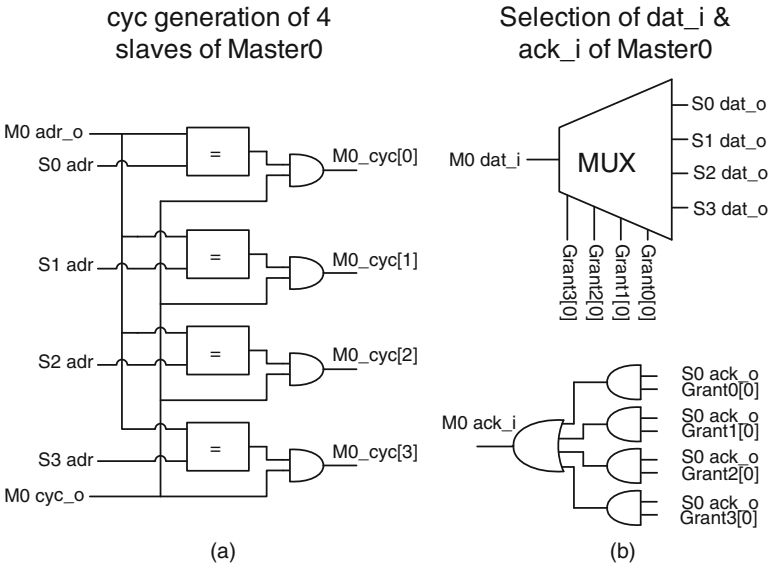

Fig. 6.8 Architecture of M0 sub-unit: (**a**) Generation of cyc signals of 4 slaves that can connect to the master, and (**b**) selection of master input signal including dat_i and *ack_i*

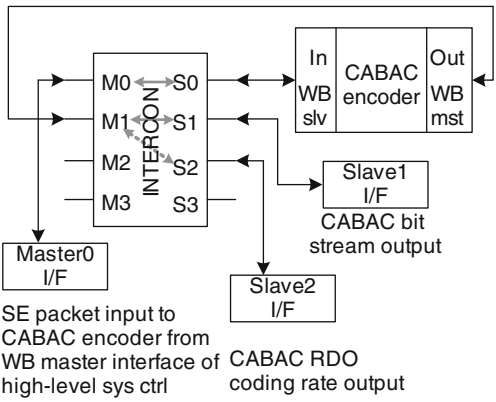

Fig. 6.9 A compact inter-connection of CABAC encoder with other components of video encoder

of INTERCON. WISHBONE slave interface of CABAC encoder is connected to the S0 sub-unit of INTERCON. An internal channel is granted from M0 to S0, which enables the input packets to transfer from Master0 to CABAC encoder. The master interface of CABAC encoder is connected to M1 of INTERCON to output packets of coded results. Based on the *adr_o* of master interface bus cycle, internal channel of M1 to S1 or M1 to S2 can be connected in INTERCON that enables packets of CABAC coded bit stream to be sent to Slave1 interface, and RDO rate to be sent to Slave2 interface. Interfaces of Master0 and Slave 2 can be located in the

host processor that controls video coding of H.264/AVC, while interface of Slave 1 can be connected to the buffer of coded frames of video encoder. Two channels of crossbar INTERCON are utilized in this compact CABAC encoding system, and no conflicts of packet input and output exist, compared to the system with shared bus INTERCON.

For the prospect of SoC-based H.264/AVC encoder based on the WISHBONE system bus, the crossbar INTERCON can be scaled up to allow more master and slave devices to connect to the WISHBONE system bus. With the accomplishment of other HW IP cores such as ME and MC, intra prediction, integer transform and quantization, in-loop deblocking filtering, etc., the IPs are connected to the INTERCON through the system bus interface. Encoding procedure of H.264/AVC video sequence can be managed in a timing-efficient manner, which enables the IP cores to encode in parallel, reduces average encoding time per MB, enhance the utilization efficiency of connection channels, and minimize the bandwidth of system bus.

Chapter 7
Circuit Design, Implementation, and Verification of CABAC Encoder

In this chapter, circuit design procedure and performance of the proposed CABAC encoder are discussed. Design and verification steps of the encoder are introduced first, followed by results discussion of synthesis and physical design. The performance of the ASIC-implemented CABAC encoder is analyzed in terms of its acceleration over CABAC software encoding, influence on the overall video encoding system performance, and the efficiency of context model access. Finally, comprehensive performance comparison is given for the proposed encoder and the existing CABAC encoder designs.

7.1 Design and Verification Flow of CABAC Encoder HW IP

In this section, design and verification flow of the CABAC encoder are introduced separately. Design steps of the proposed CABAC encoder are introduced first. Functional verification strategies of the encoder are discussed then, including verification strategies of each design step, complete verification flow of ASIC design, and FPGA prototype.

7.1.1 Steps in Designing a CABAC Encoder

Complete functions of the proposed CABAC encoder introduced in the previous 3 chapters are implemented and verified in 7 design steps, as illustrated in Fig. 7.1. The step-by-step design and verification procedure of CABAC encoder enables function verification and removes potential problems in the early stage, and reduces verification difficulty. In the encoder design procedure, functional units and blocks that generate CABAC output results are designed in earlier steps and the units or block of the additional functions are designed in later steps. The benefit is that reference coded results in RDO-off mode that are generated from SW encoder keep unchanged and verification procedure is simplified.

In Step1, binary arithmetic coding (BAC) in unit AR and BP is implemented. Context model of regular bin is provided in the input test vectors in Step1, which are

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6_7,

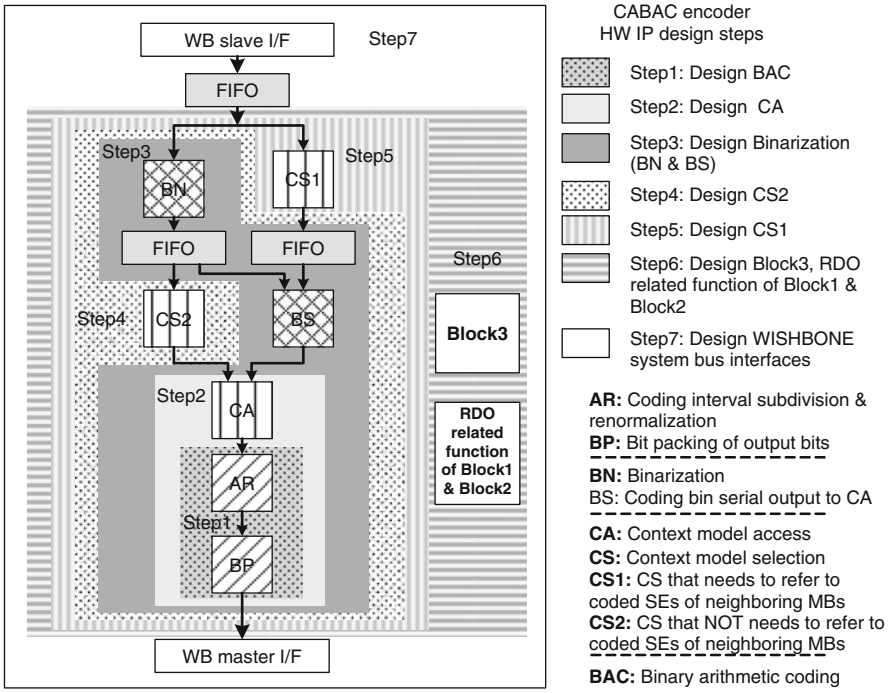

Fig. 7.1 Design steps of CABAC encoder

used to look up sub-range of LPS of coding interval. In Step2, context RAM access operations are implemented in unit CA, including read of context line that contains the accessed context model from context RAM and write of context line that contains the updated context model to context RAM. *CtxIdx* (context index) is provided in the input test vectors to address the accessed context model. The function of Block 2 is completed after Step2. In Step3, binarization of SE value and serial generation of bin packet are implemented in unit BN and unit BS, with *CtxIdx* of regular bin provided from input. In Step4 and Step5, context model selection is designed in unit CS_2 and CS_1, respectively. Unit CS_1 is the most complex unit of SE encoding with RAM access of coded SEs of neighboring blocks during selection of context model. After Step5, Block 1 is built, and CABAC encoding functions of binarization, context modeling, and binary arithmetic coding are completed. In Step6, the additional functions including context model initialization of slice initialization and context state backup and restoration of RDO-on mode coding are implemented in Block 3 and other RDO related functions are added to Block 1 and Block 2, including coding state backup and restoration of coded SEs and coding interval and RDO coding rate output. In the last step (Step7), the WISHBONE system bus interfaces with a FIFO buffer of input packets are finally designed and integrated into the CABAC encoder. This completes the circuit realization of the proposed CABAC

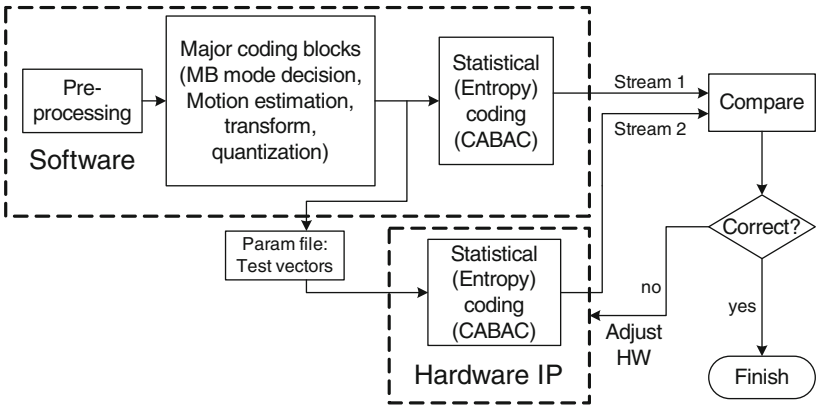

Fig. 7.2 Functional verification of the CABAC encoder block

encoder. After RTL-level function implementation in the 7 design steps, circuit synthesis and physical design are carried out targeting two types of foundry process technologies, and the results will be given in the next section.

7.1.2 Functional Verification of the CABAC Encoder

The encoder is verified using two sets of video sequences: CIF (352×288) and HDTV 720p (1,280×720). Each set of sequences contains 1 or more GOPs (Group of Pictures), and each GOP contains I, P, and B frames/fields. The encoder is tested in various video encoding configurations, including progressive/interlace coding control including frame, field, picture-level adaptive frame/field (PAFF), and MB-level adaptive frame/field (MBAFF), RDO-off/RDO-on, wide range of encoding bit rates controlled by QP, and various video encoding complexities controlled by the parameters including ME search range, number of reference frames, number of inter prediction modes, etc.

Functional verification of CABAC encoder design is illustrated in Fig. 7.2. Two bit streams are generated and compared with each other. Stream 1 is generated from the reference SW [WSJM] without any HW-assisted circuits using a video test sequence. It can contain the coded bit-stream in RDO-off mode, or RDO coding rate in RDO-on mode. Stream 2 is generated by replacing the respective CABAC SW function with the proposed CABAC encoder IP block, and feeding it with the parameter files generated by the same video test sequence. During comparison, any differences between Stream 1 and Stream 2 would indicate a design error in the CABAC HW IP.

7.1.2.1 Verification at Different Design Stages

The input to the CABAC SW encoder is redirected to a parameter file of test vectors which is then used as input to the CABAC HW encoder, assuming the same host

Table 7.1 Testing vectors of CABAC encoder at different design steps

Steps	Units	Description of input test vectors of CABAC encoder
Step1	BAC (unit AR and BP)	Bin packet containing bin, bin type, and context model
Step2	unit CA	Bin packet consisting of bin, bin type, *CtxIdx*
Step3	unit BN and BS	Packet of SE type and value, *CtxIdx* of regular bin
Step4	unit CS_2	Packet of SE type and value, *CtxIdxInc* of regular bin that needs to reference coded SE of neighboring blocks (calc by CS_1)
Step5	unit CS_1	Packet of SE type and value
Step6	Block3	Packet of SE type and value, Block3 ctrl parameters (ctx initialization, RDO ctx state backup and restoration), and RDO coding rate accumulation and output
Step7	Sys Bus I/F	*dat_i* of WISHBONE sys bus contains packet of step6

processor. For each design step discussed previously, different sets of test vectors are used. Detailed descriptions of required parameters of input test vectors at each design step are listed in Table 7.1.

In order to accelerate the verification of complex logic, excluding the reference coded results of encoder, intermediate reference results are generated for some design steps to assist verification, such as context model value in Step2, *CtxIdx* of regular bin of unit CS_2 in Step4, and *CtxIdxInc* values used to verify context model selection of unit CS_1 in Step5. One efficient verification strategy of the step-by-step encoder design is to compare intermediate coding state values of current step with previous step during verification in order to quickly locate time slot of encoding error and position of bugs.

7.1.2.2 Verification of ASIC Design Flow and Approach of FPGA Prototype

In the ASIC design flow, the encoder is tested at RTL level, gate level, and post-layout logic simulation. At the RTL level, the encoder is verified after each design step to confirm the correctness of newly implemented function unit or block. Gate-level and post-layout verifications of the encoder are also carried out to insure functional consistency utilizing netlist and SDF (*standard delay format*) file that provides timing delay information of cells and connecting wires.

FGPA prototyping is useful to find and solve design issues such as timing or area at early stage, and it is widely used in digital circuit design. Speed of FPGA verification is significantly faster compared to logic simulation. Therefore, FPGA prototyping is also attempted in this book in addition to the ASIC design to verify design prototype at the step that most functions of SE encoding are implemented including binarization, CA, BAC, and context model initialization. Xilinx ISE

[WSISE] and ModelSim [WSModelSim] are adopted to verify the encoder targeting Virtex II pro and Virtex IV FPGA at implementation steps of synthesis, mapping, and place and route. After logic simulation, bit stream of the design is generated and downloaded to the FPGA. ChipScope of Xilinx [WSCS] is utilized to insert testing probes on the design before generation of bit-stream, to monitor probe values during hardware emulation, and to transfer coded results to computer throughput JTAG cable at the pre-defined trigger condition of probes. Design is successfully verified in two test cases. In case I: testing block and CABAC encoder are implemented in Single Virtex II pro FPGA chip, and 100 MHz clock frequency is achieved. In case II, two Virtex II pro FPGAs are used for testing block and CABAC encoder. Clock frequency of case II is 20 MHz, because interconnection delay is long between two FPGA boards and interconnection wires. As the design is fully verified, the encoder can be integrated directly into FPGA video coding system.

7.2 Synthesis Results and Physical Design

The scope of this research includes logic synthesis of the SoC-based CABAC using Design Compiler [WSDC], and not the physical layout of the proposed design. The proposed CABAC encoder was laid out using TSMC 0.13 μm process technology to the GDS-II stage using Astro design tool [WSAstro]. Because of the significant differences of implemented functions of reported designs and proposed encoder, functional completeness of CABAC encoder designs are analyzed first, before the discussion on the results of circuit synthesis and physical implementation.

Most of the reported designs only implement functions of context model access (CA) and BAC or simply BAC, which are only part of functions of Block 2 of the proposed CABAC encoder, meaning the designs only complete Step1 or Step2 of the 7 steps of CABAC encoder. In comparison, the proposed design fully supports CABAC encoder function including binarization and context model selection in Block 1; context model access and BAC in Block 2; and fast context model initialization with a throughput of 4 models/cycle and fast RDO context state backup and restoration operations in Block 3. Binarization and context model selection of CABAC were not implemented in [Shojania05, Kuo06, Li06a, Nunez06], and was partially implemented in [Osorio06].

The proposed CABAC encoder is the only design that fully supports both SE encoding and RDO related operations in the HW IP. The supported RDO related functions include RDO coding rate output and 3 types of coding state backup and restoration operations for the state of context models, coded SEs of neighboring block, and coding interval of BAC. In comparison, RDO is not fully support in all the other reported designs, and the CABAC encoder [Nunez06] that focuses on HW assisted RDO only supports context state backup and restoration. In order to support CABAC encoding in both RDO-off and RDO-on modes, more complex encoding control logic is required in the proposed design to implement similar functions such as CA and BAC.

The encoding pipeline throughputs (in bin/cycle), synthesis/layout maximum operating frequencies (in cycle/s), and circuit areas (in the total number of 2-input NAND gates or mm^2) of the proposed design and other reported designs targeting CMOS process technologies or FGPA are compared in Table 7.2. In terms of areas, the proposed design is relatively larger than others. This is because the proposed design offers more complete CABAC encoder functions including context model selection and full support of RDO. Reported designs that implement similar encoder functions are compared. The proposed functional Block 3 – unique to all reported designs – takes up 8.9 K gates (20.0% of circuit area) in 0.13 μm process; while unit CS_1 of this design occupies 15.2 K gates (34.1% of circuit area) in the same technology, because large number of registers is allocated to store the coded SE data. Design of unit CS_1 and Block 3 occupy over 54% of encoder logic area, while Block 2 occupies 23.2% including RDO support in CA and BAC. Binarization and context model selection of CS_2 utilize 18.6%. Only design in [Liu07a] supports similar function of unit CS_1 using 15.9 K gates of the block. However, in [Liu07a], RDO is not supported in the block, and SE access delay of neighboring MB is significant. In terms of throughput, those of [Shojania05, Li06a, Chen07b, Liu07a] are lower than 1 bin/cycle, while those of [Kuo06, Nunez06], and the proposed design are 1 bin/cycle, and [Osorio06] is at round 2 bin/cycle. If the *maximum number of bins*

Table 7.2 Throughput, max frequency, area of CABAC encoders

Design	Process technology	Bin/cycle	Clock freq MHz	Circuit area of the implemented functions		
[Li06a]	0.35 μm syn. ROHM	0.59	150	4.57 K gates (only part of Block 2) 4.1 Kb RAM		
[Kuo06]	0.18 μm syn. TSMC	1	200	0.31 mm^2 (only part of Block 2)		
[Chen07b]	0.15 μm syn.	0.56	333	13.3 K excluding mem (*CtxIdxInc* calc. at host)		
[Liu07a]	0.13 μm syn. TSMC	0.67	200	34.3 K excluding mem		
[Nunez06]	XilinxV4 FPGA	1	130	1,158 slices (encoder only for part of block 2: 856 slices)		
[Osorio06]	0.35 μm syn. AMS	1.9~2.3	186	19.4 K excluding mem (Block 2, partial of binarization)		
[Shojania05]	0.18 μm syn. TSMC	0.33	263	0.423 mm^2 (only part of Block 2)		
	0.13 μm TSMC *post-layout*		328 (*worst-case*)	1.41 mm^2	Including: WB I/F, all RAMs, ROMs	
This design		1		Total	Unit CS_1	Block 3
	0.13 μm syn. TSMC		578	44.6 K	15.2 K	8.9 K
	0.35 μm syn. AMS		186	31.2 K	11.1 K	5.7 K

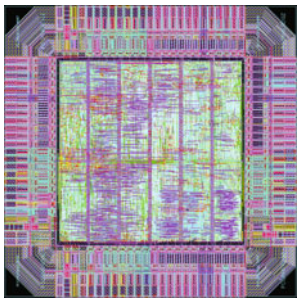

Fig. 7.3 Chip layout of the CABAC encoder

processed per second is a determining factor, the proposed design offers higher processing speed (bin/s) compared to [Shojania05, Kuo06, Li06a, Chen07b, Liu07a]. Design [Osorio06] reportedly offers the highest processing speed. Performance of the proposed encoder and [Osorio06] will be compared in details in the later section of this chapter.

The layout of the proposed design is completed with TSMC 0.13 μm CMOS process, going through design flow of floor planning, power ring and power straps insertion, placement and clock tree synthesis (CTS), routing, and DRC and LVS checking. The chip layout is shown in Fig. 7.3. The core size of encoder is 1.41 mm^2 with core utilization of 90.8%. Post-layout simulation can be run at a clock frequency of 328 MHz in the worst-case corner (1.08 V, 125°C) with constant encoding pipeline throughput of 1 bin/cycle. Up to now, this design is the only CABAC encoder reported with the most precise post-layout results.

Chapter 8
Power Reduction Strategies, MBIST, and Design Performance Comparison

8.1 Power Reduction Strategies and Power Consumption Analysis

Power consumption has always been a topic of great interest when comes to designs for portable applications. In our CABAC design, all the CABAC functions including support for RDO have been implemented with more local register buffers and larger memory allocation. Thus, it is important to implement power reduction techniques wherever possible to reduce power consumption.

RTL-level power reduction techniques are adopted in the proposed design. Net switching power of RAM blocks is reduced by decreasing RAM access frequency of both context RAMs and SE RAM with the proposed context model access scheme and MB-based SE RAM access discussed in Chap. 5.

Clock gating is applied to the whole CABAC encoder design to gate the clock signals to the registers of different function blocks and constrain cell internal power of large number of registers of encoder. During circuit synthesis, clock uncertainty is reasonably defined to estimate clock latency across clock gating cells, and hold time violations are resolved to ensure next data input of register arrives after clock edge of gated clock. Function correctness of the encoder is verified after applying power reduction techniques.

In order to precisely evaluate power consumption with the proposed designs, gate-level SAIF (Switching Activity Interchange Format) file is generated from CIF and HDTV 720p sequences simulation in both RDO-off and RDO-on modes in several test conditions. The SAIF file is imported to the state-of-the-art power analysis tool Power Compiler [WSPC]. Dynamic power of the design is significantly reduced with power reduction techniques applied.

Available power consumptions of reported designs ([Kuo06] and [Shojania05]) and proposed design (RDO-off mode) are precisely compared using same TSMC 0.18 μm CMOS process at same clock frequency of 200 MHz, as listed in Table 8.1. For the same function block of Block 2 and Normal RAM access, power consumptions of [Kuo06] and [Shojania05], and this design are 20.7, 36.5, and 19.6 mW, respectively. This design achieves power reduction of 6 and 46% compared

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6_8,

Table 8.1 Gate-level power consumption (mW) of reported designs and proposed design

Design	Total power	Power of Block 2 and normal context RAM access	
[Kuo06] (partial of Block 2 func and RAM acc)	N/A	20.7 (200 MHz, 0.18 μm)	
[Shojania05] (partial of Block 2 func and RAM acc)	N/A	48.0 (263 MHz, 0.18 μm)	36.5 (200 MHz, 0.18 μm)
This design at 200 Mbps, 0.18 μm TSMC RDO-off mode	48.9	25.7 (w RDO)	19.6 (w/o RDO)

to [Kuo06] and [Shojania05], indicating that power is more efficiently constrained compared to Kuo's low power cache-based design.

For the power consumption on the host processor, low power embedded processors typically consumes 0.2–2 mW per MIPS [Rabaey06], such as the ARM 9-11 series and the PowerPC. In the instruction-level analysis of H.264/AVC reference SW encoder [WSJM] in QP 12–28, it is found that 58.8% of CABAC computation still need to be calculated by the host processor in [Kuo06] and [Shojania05], which is an average of 1.54E+04 MIPS. Even if power-efficient embedded processor is assumed with 0.2 mW/MIPS, 3.09 W will be consumed on the host processor of reported designs, while this part of computation is entirely implemented in the proposed HW encoder. Therefore, the total power consumption of the proposed CABAC encoder is significantly lower compared to the reported designs.

Power consumption of whole encoder of the proposed design is evaluated in several video coding configurations, as shown in Table 8.2. In the RDO-off CIF test at bit rate of 980 Kbps, this design only consumes 0.087 mW at clock frequency of 1.3 MHz. To support full RDO-on coding of CIF sequence at 490 Kbps, the encoder consumes 9.95 mW at 211.5 MHz. To support HDTV 720p 60 fps RDO-off coding at 8.9 Mbps, power consumption of the encoder is 0.79 mW. Power consumption of the proposed encoder is higher in RDO-on mode because of significantly higher input SE packet frequency.

The average distribution of power consumption of the proposed CABAC encoder in different function blocks are listed in Table 8.3 for RDO-off/RDO-on modes. As shown in the table, large percentage of the encoder power is consumed by Block 1, Block 2, and memory block, while the power consumption of WISHBONE system bus interfaces is only 1.0%. Table 8.3 illustrates that percentage of power consumption of Block 3 and Memory block increases in RDO-on mode. It is because in

Table 8.2 Power consumption of the proposed encoder in 3 video coding configurations

Video format	Clock (MHz)	RDO mode	Bit rate	Power (mW)
CIF, 60 fps	1.3	Off	980 Kbps	0.087
CIF, 30 fps	211.5	On	490 Kbps	9.95
720p, 60 fps	11.6	Off	8.9 Mbps	0.79

Table 8.3 Distribution of power consumption of the proposed CABAC encoder in RDO-on/RDO-off mode coding

RDO mode	Memory (%)	FIFOs (%)	Block1 (%)	Block2 (%)	Block3 (%)	WB I/Fs (%)	Normal RAM (%)	3 func blocks (%)
RDO-off	27.0	9.1	18.9	40.4	3.4	1.0	12.3	62.6
RDO-on	28.3	8.9	20.5	36.3	4.6	1.0	11.0	61.5

RDO-on mode, Block 3 and context RAMs are more frequently accessed for context state backup and restoration of P8×8 mode decision. In RDO-on mode, unit BP (bit packing) of Block 2 consumes less power as bit packing related logic is idle, and encoding pipeline stages of unit CA and AR of Block 2 can also turn idling after bin packet processing of one RDO mode, and before high-level RDO mode decision and receiving of input packets of next mode.

The P8×8 RDO mode decision is not frequently triggered during RDO-on coding compared to some of other RDO modes such as Intra4×4, and in some situations such as HD videos with complex textures and motions, P8×8 mode does not contribute significantly to the bit rate reduction. Therefore, it is reasonable to constrain the power consumption of the P8×8 mode related function blocks. Block 3 and unit CS_1 are the two major functional blocks that control the backup and restoration of context state and state of coded SEs during P8×8 coding, and the two blocks occupy over 54% of logic area of CABAC encoder, because in unit CS_1 large local buffers are allocated to temporarily store coded SEs and 5 pipelines are utilized in Block 3. After clock gating is applied, dynamic power of unit CS_1 and Block 3 is efficiently reduced, consuming 9.5% encoder power in RDO-off mode and 11.0% in RDO-on mode.

8.2 MBIST Circuit of Memory Block of CABAC Encoder

CABAC encoder requires frequent access of context models from context RAMs and access of context model initialization parameters from ROM tables during slice initialization. Enhance testability of the Memory Block of the encoder is another design consideration. Test circuit insertion procedure of memory blocks is not similar to the scan chain insertion of registers of function logic. MBISTArchitect of Mentor Graphics [WSMBIST] is employed to generate and insert complete RTL-level Built-In-Self-Test (BIST) logic for the context RAMs and ROMs of Memory Block. After MBIST circuit insertion, the RTL-level architecture is an integration of the original coding logic together with the new test circuits required for BIST purposes. The encoder is able to either do the normal coding function or perform memory self test when activated.

A wide range of test algorithms that offer high fault coverage can be selected. March2 and ROM2 algorithms are chosen for RAM and ROM test respectively because they both cover a wide range of faults in short test procedure. The test

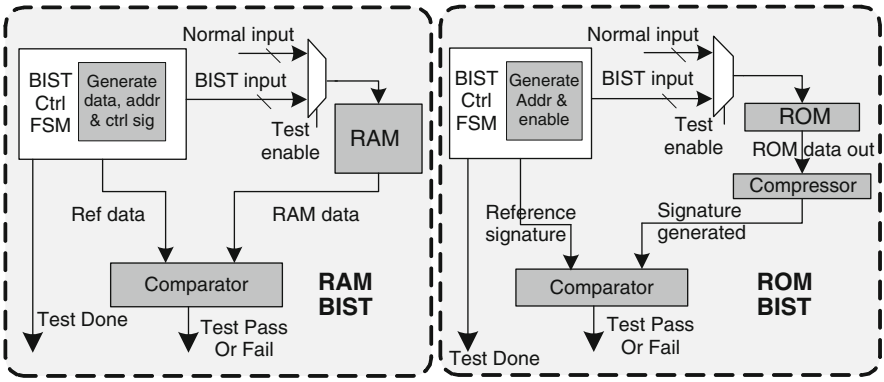

Fig. 8.1 BIST testing circuits of memory block, including RAM BIST and ROM BIST

mode is triggered when test_enable is set to 1, otherwise the CABAC encoder functions as normal. The successful completion of memory test is indicated by logic 1 in test_done and logic 0 in test_fail in the RAM test, as shown in Fig. 8.1. If any fault is found, test_fail goes high and the test ceases. RAM test can be reset at any time to initialize the test, while BIST clock uses the same clock as the encoder. For ROM test shown in Fig. 8.1, all values read from ROM are compressed into a signature by a compressor, and it is in turn compared with a reference signature when the test is done. The test fail signal goes high in the case of signature mismatch. Compared to the default scan-out result checking scheme of ROM BIST, this scheme with local comparator is simpler and more time-efficient.

In order to contain MBIST capabilities for the CABAC encoder, input BIST signals from BIST controller and the normal memory input signals are selected by the multiplexer at memory interface according to test enable signal(s). ROM BIST capabilities are added to the 4 ROM tables of context model initialization parameters, and 4 RAM BIST controllers are allocated, including 1 for Normal RAM and Temp RAM, 1 for Best RAM and P8×8 RAM, 1 for Address list, and 1 for the FIFO buffer RAM blocks. The memory blocks can be self-tested simultaneously or selectively according to the test scheme. The BIST circuits of ROMs, context RAMs, and FIFOs occupy 4.0 K gates in total in 0.13 μm CMOS process, which is not significant compared to the encoder area. Influence on the encoder clock frequency is trivial, because critical path is not located in the test logic. It is necessary to integrate MBIST and logic test circuits into the proposed encoder IP when the design is to be fabricated or integrated into a video encoding system as a hard IP in the further work.

8.3 Performance Comparison

In the following sections, performance of the proposed CABAC encoder is evaluated from the aspects of CABAC encoding acceleration, encoding throughput, and

impact of CABAC encoder IP on the overall system performance. Performance of the CABAC encoder is compared to two reported designs to illustrate the advantages of the proposed design in context model access efficiency and encoder function completeness.

8.3.1 CABAC Encoding Speed Performance of the Encoder

8.3.1.1 Acceleration of CABAC Encoding

In order to evaluate the speed up of encoding of the proposed CABAC IP, the timing performance of the proposed top-level CABAC encoding architecture is compared to that of running reference SW CABAC encoder on host processor of the video encoding system. It is done by measuring the time taken by each to perform CABAC encoding of the same test sequence. Let t_{CABAC_top} denotes the encoding time taken by the proposed top-level CABAC architecture, and t_{ref_SW} denote the time taken by the reference SW encoder, the design objective is to achieve significant time-saving as shown in (8.1).

$$t_{CABAC\_top} << t_{ref\_SW} \tag{8.1}$$

It is further assumed that the operating clock frequencies of host processor and the HW CABAC encoding architecture are similar. Because the current processor core embedded in the FPGA chip such as MicroBlaze [WSMB] in the Xilinx Virtex-4 platform can achieve clock frequency up to 200 MHz, it is reasonable to estimate that an RISC processor core can work at the clock frequency of 300~500 MHz which is similar to the clock frequency of HW CABAC encoder generated by circuits synthesis and physical design tool. Therefore, the expression for number of cycles corresponding to (8.1) can be derived as:

$$C_{CABAC\_top} << C_{ref\_SW} \tag{8.2}$$

For SW/HW co-design architecture of CABAC encoder, cycle number of top-level CABAC encoder C_{CABAC_top} consists of three parts: cycles of HW IP (C_{HW_IP}), cycles of SW non-IP (C_{SW_nonIP}) running by the host processor, and cycle of data transfer delay from host processor to HW IP ($C_{transfer\_latency}$). If encoding function of SW non-IP, HW IP, and data transfer are sequential executed, C_{CABAC_top} is the sum of three parts:

$$C_{CABAC\_top} = C_{HW\_IP} + C_{SW\_nonIP} + C_{transfer\_latency} \tag{8.3}$$

C_{SW_nonIP} of the proposed design is minimized compared to all reported designs, because CABAC encoding functions are full implemented in HW IP, and the host processor only needs to re-package the input SE packets and control parameters. C_{SW_nonIP} is also significantly lower compared to C_{HW_IP}. Moreover, HW IP and SW non-IP are processing CABAC HW input packets in parallel, and

Table 8.4 Speed-up of CABAC encoding of the HW IP compared to SW

RDO mode	QP	24	28	32
RDO-off	C_{HW_IP}	3.2E+05	1.9E+05	1.2E+05
	C_{ref_SW}	4.6E+07	2.8E+07	1.8E+07
	Speed-up	145	147	153
RDO-on	C_{HW_IP}	8.0E+07	6.3E+07	5.2E+07
	C_{ref_SW}	2.1E+09	1.6E+09	1.2E+09
	Speed-up	26	25	24

C_{HW_IP} is significant larger compared to C_{SW_nonIP}. Thus, C_{SW_nonIP} and transfer delay $C_{transfer\_latency}$ can be ignored in (8.3), and C_{CABAC_top} can be represented by C_{HW_IP} for the proposed top-level CABAC encoding architecture. The acceleration of CABAC encoding is illustrated in Table 8.4, in which number of cycles of HW IP, cycles of reference SW encoding, and acceleration times of the HW IP are listed for coding test of one GOP CIF format Foreman sequence at 3 QP values.

In RDO-off mode coding, speed-up of HW CABAC encoder are significant in different QPs, and it is more efficient in high QP (low bit rate). It is because the percentage of regular bins is proportional, and SW encoding of regular bin is inefficient, which requires memory access in multiple cycles. In RDO-on mode, speed-up is relatively lower compared to RDO-off mode. This is because in RDO-on coding, CABAC encoding pipeline is empty when coding rate is output when all bins of current mode are encoded, and the multi-stage pipeline is reloaded when the mode decision is made, and SE packets of next RDO mode are received by the encoder. Pipeline efficiency is degraded; however, the encoding speed of proposed encoder is still faster in RDO-on mode. Speed-up can be further increased if the RDO mode decision algorithm of host processor is adjusted. SE packets of the next RDO mode can be generated before the coding rate of current mode is calculated in the encoder IP, if the decision of next RDO mode does not depend on the coding rate of current mode, such as intra prediction mode decision of 4×4 blocks.

8.3.1.2 Constant Throughput of CABAC Encoder

Although encoding throughput is limited by the maximum throughput of BAC pipeline stages, it is also influenced by the performance of previous encoding stages. For the full HW CABAC encoder design, SEs of residual blocks are generated during binarization. Bin strings of one residual block cannot be generated until all values of residual coefficients are received. Therefore, bubbles can be inserted in the pipeline when the bin string of next SE is not available. Inaddition, context model selection is also a complex procedure when the coded SEs of neighboring blocks needed be referenced and accessed from SE RAM, in which situation pipeline can be stalled when the *CtxIdxInc* calculation of regular bin cannot be completed when *CtxIdx* is needed. In this book, the problem of residual SE generation is released by (a) input run-level pairs instead of all coefficients of residual block to reduce

Table 8.5 Average throughput of the proposed CABAC encoder in video coding tests

QP	12	16	20	24	28	32
Total encoding cycles	1,874,123	1,103,743	566,390	314,376	190,255	117,383
Bin number	1,855,991	1,096,048	562,211	311,756	188,186	115,195
Avg throughput	0.990	0.993	0.993	0.992	0.990	0.981

residual input cycles, (b) calculate SEs during run-level pair input procedure, and (c) insert FIFO buffer after unit BN. Context model selection is also accelerated by partitioning the function in unit CS_1 and CS_2, the pre-calculation of *CtxIdxInc* of unit CS_1 in parallel with binarization of BN. Average throughput of the entire encoder is tested in one GOP of CIF Foreman sequence in the QP range of 12–32. As shown in Table 8.5, throughput of the encoder is constant in the whole QP range. Pipeline stall probability is minimized to 1% or even lower, and the stall is only caused when FIFO of bin string is empty and SEs of next residual block are still being processed in unit BN. The average throughput of encoder is 0.99 bin/cycle, which is quite approximate to the maximum throughput of 1 bin/cycle of CABAC encoding pipeline. In comparison, actual throughput of [Liu07a] and [Osorio06] are degraded, compared to the reported maximum throughput, as design of binarization and context model selection is inefficient and encoding pipeline stall frequency is high. It will be further discussed in Sect. 8.3.3.

8.3.1.3 Worst-Case Analysis of Proposed CABAC Encoder

For the worst-case analysis, the H.264/AVC encoder is tested in HDTV 720p 60 fps coding, with all encoding tools turned on including full RDO mode. The best coding performance is achieved at the cost of high computation. It is found that to achieve output bit rate of 10.7 ~ 24.4 Mbps, CABAC encoder needs to support encoding speed of 0.77 ~ 1.52 Gbin/s in RDO-on mode, which is beyond the processing speed of any of the CABAC encoder designs at current stage. For the proposed CABAC encoder at post-layout stage, a maximum encoding speed of 325 Mbin/s at worst-case corner (1.08 V, 125°C) is supported. However, in the typical case, the post-layout design can work at significantly higher encoding speed of over 500 Mbin/s. To achieve HDTV real-time RDO coding, simplified RDO-on mode may be adopted to reduce RDO testing modes, and the scheme of parallel encoding of multiple slices of same frame on multiple CABAC encoding engines may be applied.

8.3.2 Performance Comparison of Context Model Access Efficiency

The performance of context model access of the proposed design is evaluated from two aspects: (1) efficiency of context RAM read and write access; and (2) efficiency of context state backup and restoration operation. The performance is compared

with [Nunez06], which is the only reported design that fully supports different RDO context state backup and restoration operations. The influence of context RAM reallocation is also evaluated.

8.3.2.1 Comparison of Context RAM Read and Write Access Efficiency

CABAC encoder is tested using 4 CIF video sequences: Foreman, Mother and Daughter (M&D), Coastguard, and News, with QPs set at 12, 16, 20, 24, 28, and 32 in both RDO-off and RDO-on modes. The context RAM read and write access frequency ratio of this design over [Nunez06] is illustrated in different coding conditions.

- RDO-off mode

  In the RDO-off mode coding test of IPB video sequence (2 GOP, 19 CIF frames), the averaged context RAM access frequency ratio of this design over [Nunez06] (*access ratio*) in the QP range of 12–32 is shown in Table 8.6. With techniques of both context line access scheme and context RAM reallocation applied, the context RAM access frequency of this design is 26.1–33.2% of that of [Nunez06] in the 4-sequence tests, with an average of 28.4%. The RAM access frequency of this design is significantly lower than [Nunez06].

  Applying the context line access scheme alone, the average access ratio of this design over [Nunez06] is 33.9%. After context RAM reallocation, the average RAM access frequency of this design further reduces 16.1%, as shown in Table 8.7, indicating that the reallocated context RAM efficiently reduces context RAM access frequency.

  Figure 8.2 illustrates the access ratio with respect to QP. In the low QP range, RAM access ratio of sequence Coastguard is lower (more efficient), while access ratios of M&D and News are higher. However, in the high QP range, the access ratio of Coastguard increases rapidly. For sequence News, variation of access ratio is small in the whole QP range.

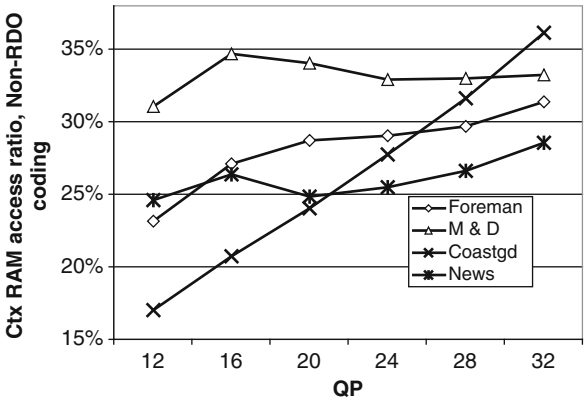

Fig. 8.2 Context RAM access frequency ratio of this design over [Nunez06], during RDO-off coding in the QP range of 12–32 of 4 typical video sequences

The access ratio is more sensitive to the change of QP if the total energy of transform coefficients is more scattered in frequency domain (low and high frequency coefficients), such as Coastguard. In comparison, if the energy of transform coefficients concentrates more in the low frequency, such as News, the access ratio is more stable as QP changes, compared to that of Coastguard.
In the low QP range, context RAM access of high motion sequences Coastguard is more efficient with lower access ratio, while RAM access ratio of low motion sequence M&D and News is higher. This is because higher motion of video sequence introduces larger MVD and residual blocks with more non-zero coefficient after quantization, which reduce the context RAM access ratio. However, in high QP range, access ratios of high motion sequences increase more rapidly. At QP 32, access ratio of Coastguard is highest among all sequences. In comparison, low motion sequence News has lower access ratio. As QP increases, bit rate of Coastguard reduces more rapidly compared to that of M&D and News. As motion estimation related SEs need to be processed in all QP range, variation of RAM access ratio and bit rate is caused by the change of quantized residual coefficient. Prediction error energy of Coastguard is more evenly distributed in low and high frequency range, and less energy is kept as most coefficients are quantized to 0 in high QP coding, and access ratio increases rapidly. Residual coefficients energy of News is more concentrated in low frequency range, and more energy is kept and more non-zero coefficients are kept after high QP quantization. Therefore, context RAM access ratio of News is lower in high QP.

- RDO-on mode

Table 8.6 Average context RAM access frequency ratio (This design over [Nunez06] in RDO-off mode coding)

Sequence	RAM access ratio (%)
Foreman	28.2
M&D	33.2
Coastguard	26.2
News	26.1
Average	28.4

Table 8.7 Reduction of RAM access frequency of the proposed encoder, attributed to Context RAM reallocation

Sequence	RAM access reduction (%)
Foreman	13.7
M&D	14.6
Coastguard	18.8
News	17.5
Average	16.1

The variation of RAM access frequency of P or B frames in the same GOP is very small in the video coding tests. Because computation complexity increases significantly during RDO coding, proposed design is tested with the 1st I, P, and B frame in RDO-on mode.

Figure 8.3 shows the average RAM read frequency ratio (read ratio) and RAM write frequency ratio (write ratio) of this design over [Nunez06] in the QP of 12–32. Average read ratios are 22.4% (I), 23.9% (P), and 24.1% (B), and write ratios are 14.5% (I), 15.3% (P), and 15.4% (B). RAM access of News is more efficient compared to that of other sequences. For each sequence, the read and write ratios of P and B frames are higher than I frames, because the percentage of non-residual SE in the P or B frame is higher, and RAM access efficiency of non-residual SE is lower than that of residual SE. Compared to RDO-off coding, context access of this design is more efficient (lower access ratio) in RDO-on coding because large percentage of RDO modes are Intra prediction mode decision of 4×4 block (Intra4×4), in which the percentage of residual SEs is higher. In RDO-on coding, write ratio is significantly lower than read ratio, because in non-P8×8 RDO mode coding, the last two context lines stored in the context line buffers of unit CA need not to be written back to Temp RAM. The difference of read and write ratio is more significant for the low bit rate sequence, such as M&D.

Figure 8.4 shows the read ratio and write ratio of this design over [Nunez06] with respect to QP in I, P, and B frame RDO coding. The RAM access ratio increases with QP because the percentage of residual SE decreases. The curves of inter (P and B) and intra frame (I) are similar, because most RDO coding modes of inter frames are still Intra4×4 modes in inter coding, which is a dominating factor of RAM access efficiency. Spatial characteristics of the frame have stronger influence on the RAM access property than the temporal characteristics.

In Intra4×4 mode RDO coding, 4 types of residual SEs are processed in high ratio: CBF, SCF, LSCF, and abs_level_minus1. Several factors are found that influence the RAM read ratio for residual SE coding. (a) The sum of abs level values in the coding 4×4 block *blk_sum*: decides the utilization efficiency of context models of abs_level_minus1; (b) Non-zero coefficient number in each block *blk_coef_num*: influences the access of LSCF context models; (c) Percentage of 4×4 blocks with zero coefficients in the total blocks processed *zero_blk*: If the percentage of zero blocks is high, utilization of context line is less efficient; (d) Position of the Last SCF in the block: decides average number of SCF context models accessed per block. These factors reflect the distribution of residual transform coefficients energy and influence the efficiency of RAM read access in RDO-on mode coding.

Power consumption of the context RAM during RDO-on and RDO-off coding is also compared with reference design [Nunez06] and other designs with single context model access scheme. The widely used industry SRAM model CACTI 5.3 of HP lab [WSCACTI] (Available from: quid.hpl.hp.com:9081/cacti/sram.y) is utilized to evaluate SRAM access energy of the two types of SRAM architecture including

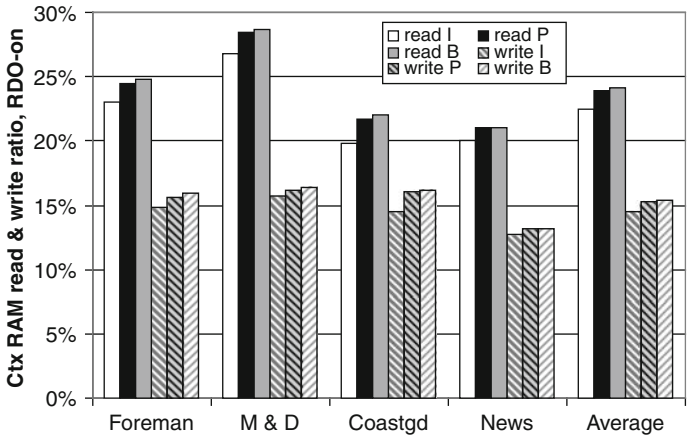

Fig. 8.3 Context RAM read and write frequency access ratio of this design over [Nunez06], during RDO-on coding. The average access ratios of I, P, and B frames of 4 video sequences in QP range of 12–32 are shown

53-word × 56-bit of this design and 400-word × 8-bit of reference. According to SRAM access frequency ratio of this design and [Nunez06], context RAM power consumption is evaluated. Power reduction of context RAM of this design is 16% in RDO-off, and 31% (read) and 56% (write) in RDO-on mode compared to the reference.

8.3.2.2 Efficiency of Context State Backup and Restoration (B&R) Operation

Compared to [Nunez06], the operations of context state restoration are removed during non-P8×8 RDO coding, because of allocation of Temp RAM to store updated context lines during coding of each RDO mode, as discussed in Chap. 5. During RDO-on coding of P8×8 sub-MB mode decision, 4 partition modes 8×8, 8×4, 4×8, and 4×4 are tested for each 8×8 sub-MB. The context state of context RAM needs to be restored after coding of each RDO mode, and the context state of the best mode needs to be stored to context RAM after all modes are tested. With the architecture of 3 backup FIFO buffers and 1 coding context RAM, all context state restoration operations must be executed in [Nunez06], and at least 20 context state B&R operations are required per inter MB.

In the proposed design, the backup and restoration operation is removed if the mode is not the currently best mode. With pipeline structure in the proposed design, the backup and restoration operations can be executed concurrently, so at most 1 operation is performed per mode. In each operation, because of context line are accessed, the average cycles per operation are also lower than [Nunez06]. With fewer cycles per operation and fewer operations per MB, the operation timing delay per MB (cycle number) of this design is significantly lower than [Nunez06].

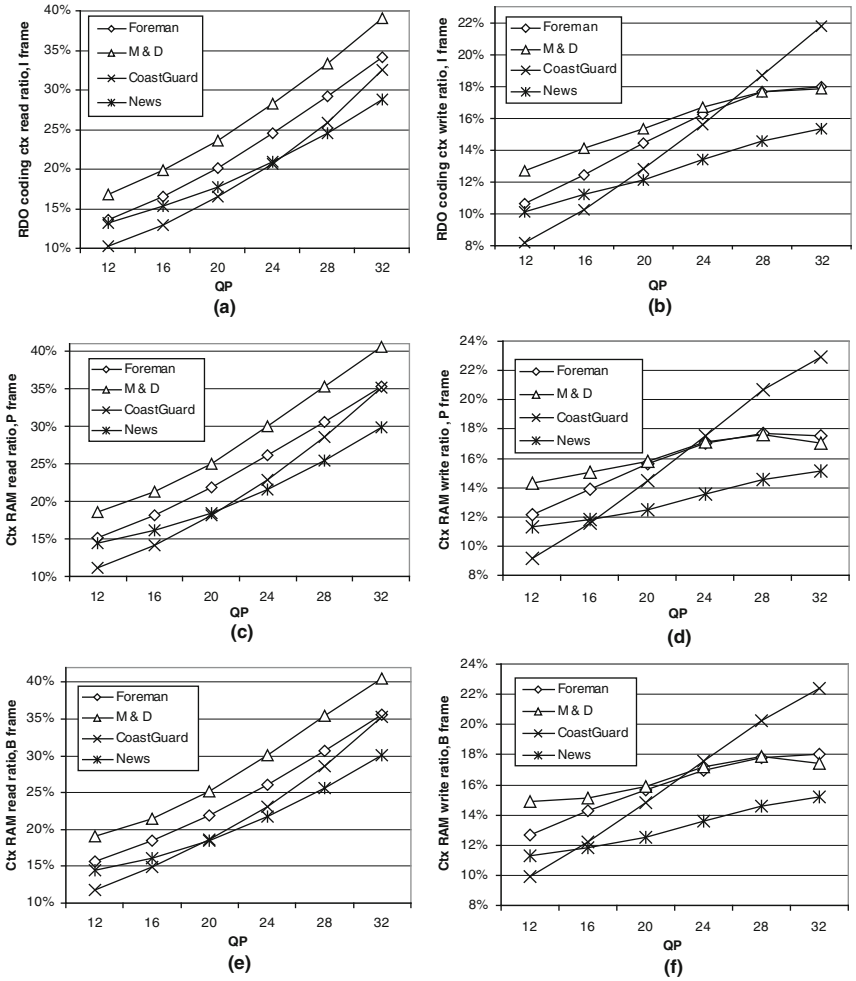

Fig. 8.4 Context RAM access frequency ratio of this design over [Nunez06] during RDO coding in the QP range of 12–32 of 4 video coding sequences. Read ratio of I, P, and B frames are illustrated in (**a**), (**c**), and (**e**) respectively; Write ratio of I, P, and B frames are illustrated in (**b**), (**d**), and (**f**)

Average P8×8 context state B&R operation delay ratio *ctx_opt_delay_ratio* of the proposed encoder to [Nunez06] is evaluated by (8.4). *MB_num_frm* is the number of MB per frame. *ctx_opt_frm* is the average number of context state B&R operations taken in each frame, which can be obtained in video coding tests. In each context state B&R operation, ratio of operation cycles of proposed design to [Nunez06] is evaluated as the ratio of the context lines accessed in proposed design (*ctx_line_num*) to the context models accessed in [Nunez06] (*ctx_model_num*) during P8×8 sub-MB mode decision. Average numbers of *ctx_line_num* and *ctx_model_num* are used for the calculation of (8.4).

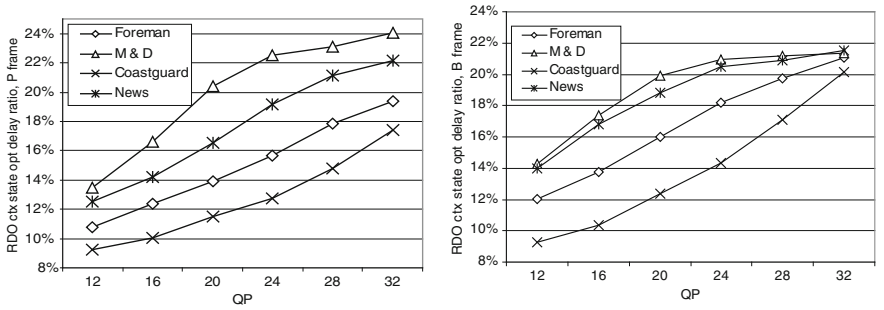

Fig. 8.5 Context state backup and restoration operation delay ratio of this design to [Nunez06] in P8×8 RDO coding for QP 12–32 of 4 video coding sequences. Ratio of P frame coding in (**a**) and ratio of B frame in (**b**)

$$ctx\_opt\_delay\_ratio = \frac{ctx\_opt\_frm}{20 \times MB\_num\_frm} \times \frac{ctx\_line\_num}{ctx\_\ model\_num} \tag{8.4}$$

Figure 8.5 illustrates the timing delay ratio of this design to [Nunez06] of context state B&R operation in P8×8 RDO coding. Operation delay ratio is lower (more efficient) in high motion sequence such as Coastguard, and higher in lower motion sequence incluidng M&D and News. Because only 2 context lines are accessed to encode a MVD, context RAM access efficiency is higher for larger MVD. For high motion sequence, the percentage of non-zero MVD and residual coefficients are larger compared to that of low motion sequence. The ratio of *ctx_line_num* to *ctx_model_num* in (8.4) decreases when the percentage of non-zero MVD increases. Increase of non-zero residual coefficients also causes decrease of the delay ratio. The variation of *ctx_opt_frm* in each sequence is not as significant as that of the ratio of *ctx_line_num* to *ctx_model_num*. In general, the context state backup and restoration operation of proposed CABAC encoder is more timing efficient for high motion sequence such as Coastguard. The average operation delay of this design is 15.5 and 16.6% of [Nunez06] for P and B frame coding of in 5 CIF sequences coding tests, as shown in Table 8.8.

Table 8.8 Average context state backup and restore operation delay ratio of the proposed design to [Nunez06]

Sequence	P frame (%)	B frame (%)
Foreman	15.0	16.8
Walk	12.3	14.4
M&D	20.0	19.2
Coastguard	12.6	13.9
News	17.6	18.8
Average	15.5	16.6

8.3.2.3 Context RAM Occupation Reduction

In [Nunez06], 46 Kbits backup context memory is allocated to support RDO coding. Instead of allocating 3 large FIFOs to backup 3 intermediate context states during P8×8 RDO coding, only 4 small RAMs are allocated to support RDO, including Temp RAM, Best RAM, P8×8 RAM, and Address list, and Best RAM and P8×8 RAM are used to backup only maximum of 11 context lines that can be modified during P8×8 mode decision. Inaddition, because context models are accessed by context line instead of single context model in the proposed design, fewer memory addresses of the accessed context models need to be stored in the backup RAMs. Compared to [Nunez06], only 7.37 Kbits of context RAMs are allocated in this design, which is 16.0% of [Nunez06]. The reduction of context memory size in the proposed design is attributed to the context line access scheme and more efficient allocation of backup memory resources.

8.3.3 Performance Comparison with the State-of-the-Art CABAC Encoder Design

8.3.3.1 Function Completeness

Design [Osorio06] is reported to have higher coding throughput of 1.9~2.3 bin/cycle because it was claimed that pairs of bins (combinations of regular bins and/or bypass bins) can be processed per cycle. However, the throughput is only evaluated at the pipeline stages of BAC based on the statistical distribution of generated bin pairs of test video sequences, without consideration of the processing capability of previous encoding stages including binarization and bin packet generation. The claimed throughput can only be achieved on the condition that bin pairs and the related context models are continuously fed to the BAC of encoder without any stall, which is not possible for the proposed encoder architecture. Several functional differences of [Osorio06] and the proposed design are summarized in Table 8.9.

Table 8.9 Functional comparisons of [Osorio06] and the proposed design

Function difference	[Osorio06]	This design
Binarization for non-residual SE	No	Yes
CtxIdxInc calculation (context model selection) for regular bins	Only support for SCF, LSCF, level	Full support for all types of SE
Additional bin pairing process for non-residual SE in the host processor	Yes	No
Input of residual coefficients per 4×4 residual block	16 cycles, degrade actual throughput	1~16 cycles, lower input delay
Context model initialization	No	Yes
P8×8 RDO coding modes	No	Yes

For non-residual SE and CBF processing of [Osorio06], binarization, bin pair preparation, and context model selection are assumed to perform by the host processor. As discussed in the instruction-level analysis of CABAC encoding of Chap. 4, these operations take up from 23.5 to 47.2% of total CABAC encoding instructions in CIF format coding. To enable data pairing operation in [Osorio06], computation cost on the host processor is even higher, and CABAC encoding speed is limited by the speed of host processor. In comparison, these operations are moved to the CABAC hardware encoder in the proposed design, and the host processor is mainly used to send output SE packets.

In this design, coefficients of 4×4 residual block generated in host processor are sent to CABAC encoder by run-level pairs. It reduces the average input delay of residual block compared to [Osorio06], especially in high QP coding, in which the ratio of non-zero residual coefficients are low. In [Osorio06], 16 cycles are required to input each residual 4×4 block regardless of block types and coding QP, which is significantly longer than that of the proposed design in high QP coding. In [Osorio06], the encoder stalls and the throughput is degraded when the ratio of zero coefficients in the residual block is high, because not enough residual bin pairs can be generated for the BAC stage. This situation does not occur in the proposed design. Design [Osorio06] can support non-P8×8 RDO coding mode. However, the support of P8×8 RDO coding is critical to the accuracy of motion estimation of H.264/AVC. Compared to [Osorio06], the proposed design supports complete SE encoding function including context model selection, provides complete supports of RDO including P8×8 RDO coding, and reduces computation on the host processor and bandwidth of system bus to the minimum.

8.3.3.2 Context RAM Access Efficiency

Design [Osorio06] allocates a small cache of 4-line×4-context model to reduce RAM access delay. It focuses on the efficient context access scheme for the residual SEs. Thus, the context access performance of residual SEs of [Osorio06] and this paper can be compared. In Fig. 8.6, average numbers of context RAM access for each CIF frame coding in the GOP of I, P, and B of the two designs are compared. Although 16 context models are buffered in both designs, the context RAM access efficiency differs in various QP ranges and in RDO-off mode (Fig. 8.6a) and RDO-on mode (Fig. 8.6b), and the context RAM access numbers of two designs in QP 12–28 are listed in Table 8.10.

In RDO-off coding, the averaged RAM access efficiency of [Osorio06] is a little better (with lower access number) in the low bit-rate range (high QP from 20 to 32) for one GOP coding in the whole QP range. It is because two cache lines are allocated to store the context models of CBF, and the 1st encoding level coefficient, and the RAM access of these two types of SE are reduced. However, in the high bit-rate range (low QP from 12 to 20), the RAM access performance of the proposed design is better, because it is more efficient to access context models of significant map, and large level values using wider context line of 8 models instead of 4 models per cache line in [Osorio06]. Because this CABAC design is targeting high quality

and high bit-rate coding, it is beneficial to have lower RAM access frequency in this QP range. Table 8.10 also shows that average context RAM access efficiency of the proposed design is better (7.1% reduction) for intra frame coding in RDO-off mode. It is because percentage of non-zero residual coefficients of residual block is larger in intra frame coding, and it is beneficial to use large context line to reduce RAM access of context models significant map and levels.

In full RDO-on mode, the context RAM access frequency of the proposed design is significantly lower (better) in QP range, as shown in Fig. 8.6b and Table 8.10. It is because in the RDO-on mode of [Osorio06], the two cache lines containing context models of CBF and 1st level coefficient need to be fetched from RAM to restore the context state after coding of each RDO mode, which causes additional RAM accesses. On the average of one GOP coding, 27.6% of context RAM access operations are reduced compared to [Osorio06] in RDO-on coding, and 28.7% of access can be reduced for intra frame coding.

In general, small cache based context access scheme of [Osorio06] is more efficient in low bit-rate RDO-off CABAC coding, while the proposed context access scheme with larger context line buffers is more beneficial in high bit-rate coding and RDO-on. Operation delay of context state backup and restoration in P8×8 RDO coding and related context RAM size are not comparable because the operation is not supported in [Osorio06].

8.3.3.3 SoC-Based CABAC Encoder IP

Compared to [Osorio06] and most other reported designs, system bus interfaces (WISHBONE compatible) are integrated in the proposed CABAC encoder to improve the portability and reusability of the encoder in the further SoC-based video codec systems. The data transfer rate at WISHBONE system bus interfaces of the proposed design is related to the CABAC coding mode, video resolution, frame

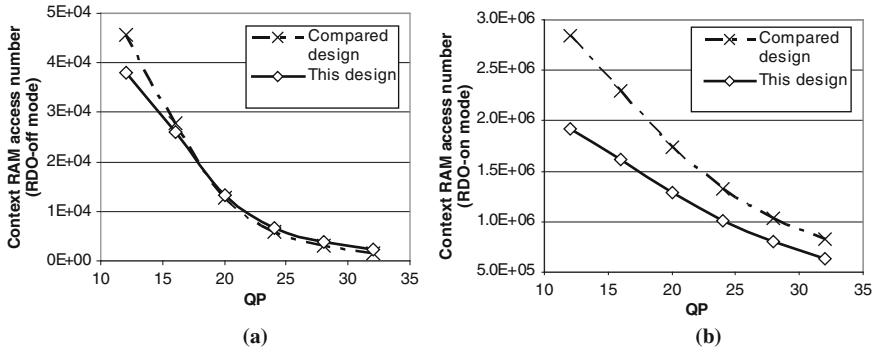

Fig. 8.6 Average context RAM access number per frame of residual SEs in [Osorio06] (compared design) and this design in CIF frame coding for QP 12–32. The access numbers of RDO-off coding and RDO-on coding are shown in (**a**) and (**b**), respectively

Table 8.10 Context access performance (number of RAM access) of the proposed encoder compared to [Osorio06] in residual SE coding

Test cond.	RDO	Design	QP 12	QP 16	QP 20	QP 24	QP 28	Avg. %↓
Avg. of 1 frame of one GOP	Off	This	38,026	26,015	13,283	6,757	3,725	
		[Osorio06]	45,542	27,788	12,796	5,932	3,059	
		%↓	16.5	6.4	–3.8	–13.9	–21.7	–3.3
	On	This	1,919,744	1,620,159	1,287,209	1,015,775	809,390	
		[Osorio06]	2,840,789	2,299,346	1,743,681	1,323,572	1,039,464	
		%↓	32.4	29.5	26.2	23.3	22.1	26.7
First intra frame	Off	This	48,754	40,412	31,237	21,934	15,322	
		[Osorio06]	62,428	48,178	33,743	21,697	14,086	
		%↓	21.9	16.1	7.4	–1.1	–8.8	7.1
	On	This	1,754,173	1,549,247	1,305,428	1,067,836	873,608	
		[Osorio06]	2,701,221	2,293,064	1,828,704	1,419,576	1,126,159	
		% ↓	35.1	32.4	28.6	24.8	22.4	28.7

This: The proposed encoder of this book.
%↓: Percentage of encoding cycles reduction, proposed design compared to [Osorio06].

rate, and compression ratio. In video coding test of the proposed encoder, to support RDO-off coding at 16 Mbps bit rate, input data rate to the encoder is 5.8 M packet/s, and output data rate is 1.9 M packet/s. Data transfer on the system bus and data packet output rate from the host are very low. Output bit packets can be sent to destination memory space directly with very low-complexity control from the host processor. To support full RDO-on mode coding of the same video sequence at 328 MHz post-layout clock frequency, input data rate is 97.3 M packet/s, and output RDO rate is 11.3 M packet/s.

Chapter 9
Conclusions

In this book, the entropy coding tools adopted by the H.264/AVC video compression standard are presented in both algorithms and VLSI architectures.

In part A, introduction to the background on video compression, including entropy compression is provided. Introduction to the CAVLC and CABAC algorithms and their respective realized VLSI architectures are also discussed in details in part A.

In part B, design and implementation of a high-performance CABAC encoder targeting the Main profile of H.264/AVC video coding standard has been conducted. The main purpose of this work is to accelerate the serial CABAC encoding, and remove the bottleneck of the H.264/AVC entropy encoding by hardware IP (intellectual property) design of CABAC encoder, which can support the following important design features: functional completeness of CABAC encoding of SE, high coding throughput, SoC-based IP design with enhanced reusability and portability, complete functional support of rate-distortion optimization (RDO), efficient context model access, and low power consumption.

The major design advantages of the proposed CABAC encoder IP and contributions are summarized as follows.

1. *Full-hardware CABAC Encoding of Syntax Elements (SEs)*
   Compared to most reported designs, the proposed CABAC encoder fully supports all 3 steps of SE encoding: binarization, context modeling, and binary arithmetic coding (BAC). The most complex CABAC encoding algorithm – context model selection of context modeling is completely designed in hardware that supports single cycle selection of all categories of context models. The benefits of full-hardware SE encoding include: (1) computation complexity on the host processor of the video codec system is significantly reduced for the preparation of CABAC input data, and the data transfer bandwidth on the system bus is also reduced; (2) the bottleneck of CABAC encoder data input is removed, because when context model selection is calculated on the host processor HW encoding is paused and throughput of the top-level CABAC encoder is degraded; (3) reusability of the hardware CABAC encoder IP is enhanced and integration complexity of the encoder in high-level video codec system is significantly reduced.

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6_9,

2. *Complete Solution of CABAC Encoding in RDO Coding Modes*
   Compared to all reported designs, the proposed CABAC encoder is the only design (to date) that solves the difficulties to develop RDO functions in CABAC encoder design with the approaches of allocating small RAMs to record intermediate context states, pipelined context state backup and restoration, multiplexing of bit backing and RDO rate accumulation, local buffering of best mode SE states, etc. It fully supports RDO related functions, including RDO coding rate generation and elaborate operations of backup and restoration of CABAC coding states including state of context models, state of coding interval of BAC, and state of coded SEs for the context model selection. RDO is one of the key techniques that enhances the coding efficiency of H.264/AVC, and support of RDO is significant to the CABAC encoder design to further expand the application fields of the design, because RDO is indispensable in the high-quality high-definition video coding applications such as HDTV and movie studio encoding systems.
3. *High and Constant Encoding Speed*
   Several design strategies are explored and applied in the proposed encoder to reduce data dependency of CABAC encoding steps and improve encoding speed (in terms of throughput $\times$ clock frequency) of the CABAC encoder. Full pipelined architecture of bin encoding is designed that can process 1 bin packet/cycle. Functional partitioning of encoding pipeline stages of bin packet generation, context model access, coding interval subdivision and renormalization, and bit packing efficiently removes the data dependence of the three sequential coding steps: (1) access of context models from context RAM, (2) binary arithmetic coding, and (3) bit packing and bit stream output. Insertion of FIFO buffers in the encoding flow enables parallel processing of SE binarization, complex context model selection, and bin encoding pipeline, and efficiently removes pipeline bubbles of CABAC encoding. With the adoption of FIFO buffer insertion and pipelined coding structure, constant encoding throughput is ensured in different video coding configurations, compared to the variable throughput of [Osorio06] and [Liu07a]. In addition, circuit critical path is significantly reduced and higher encoder clock frequency is achieved, compared to most reported designs. The encoding pipeline throughput of 1 bin/cycle and post layout circuit clock frequency 328 MHz using 0.13 μm TSMC process is suitable for HDTV real time encoding in RDO-off mode, and CIF real time encoding in RDO-on mode.
4. *Efficient Context Model Access*
   An efficient context model access scheme of CABAC encoder is also proposed in this book, including techniques of context line access and buffering, context memory reallocation, and pipelined context model B&R operation in P8$\times$8 RDO coding. Context memory size is reduced to 16.0% of [Nunez06]. Context RAM read and write access frequency in both RDO-off and RDO-on coding modes are significantly lower than [Nunez06]. Context state backup and restoration operation delay of P8$\times$8 RDO coding mode is 15.5 and 16.6% of [Nunez06] in P and B frame coding tests, while the operation of non-P8$\times$8 coding is saved. With the reduction of memory access frequency, power consumption of context

RAM blocks is also reduced. Compared to the cache-based context model access of [Osorio06], context model access frequency of the proposed design is significantly lower in RDO-on mode, and the timing delay of cache miss data fetch is avoided.

5. *Low Power Encoder Design*
   Compared to most reported designs, the proposed encoder is a low power design with power reduction techniques applied including clock-gating and context RAM access frequency reduction. Power consumption of HW CABAC encoder is efficiently constrained and total power consumption of CABAC encoding on the host processor and HW encoder is significantly lower compared to the reported designs. Therefore, proposed encoder is also suitable for portable and mobile applications, in which power consumption is a critical design consideration.
6. *Other Advantages of the Proposed CABAC Encoder*
   The proposed encoder achieves fastest context model initialization with processing throughput of 4 context models per cycle during slice initialization. Lowest operation delay of slice initialization is achieved compared to reported designs, which can be attributed to the parallel and pipelined circuit architecture. MBIST circuit insertion is also attempted for the context RAMs and ROMs of the encoder with simplified interface testing signals and self-test procedure, which can be applied in the further system integration and ASIC fabrication procedures to enhance testability of the proposed IP.

To summarize, a full-hardware high-performance low power SoC-based CABAC encoder IP is designed in this book utilizing different design strategies to achieve complete function features, high and constant coding throughput, and improved reusability and portability. This design is verified, synthesized, and laid out at the GDS-II stage, and post-layout speed is suitable for the video applications of real time CIF coding in RDO-on mode and HDTV coding in RDO-off mode.

Several design strategies utilized for the proposed CABAC encoder of this book can be further utilized in the similar R&D projects. The strategies include:

- Widely used pipeline architectures in the operations of bin encoding, context state backup and restoration, and context model initialization that enhance data processing throughput and reduce operation delay.
- Strategies of data prefetch and pre-calculation to reduce data dependency, operation delay, and critical path length, such as prefetch of context model from context RAM, pre-calculation of possible values of $\text{Range}_{\text{LPS}}$ and pre-calculation of context model selection that require access of coded SEs of neighboring blocks.
- Reduction of RAM access frequency for the operations that require frequent memory access, utilizing design strategies including context line access and local buffering, context memory reallocation, etc.
- Strategy of proper top-level functional partitioning with FIFO buffer insertion that enables parallel data processing of original sequential coding stages.

The design strategies utilized in the proposed CABAC encoder can be referenced in the designs that are of serial data processing nature, require frequent memory access, or have strong data dependency, such as statistical codec designs including CABAC decoder and CAVLC codec of H.264/AVC and statistical (entropy) codec design of JPEG2000, or other similar data processing codec designs.

Although the proposed CABAC encoder is designed targeting the Main profile of H.264/AVC standard, it can be easily scaled to the higher profiles. Additional area is required for the memory storage of context models and control logic circuits for encoding of 8×8 transform coefficients. Similar design architectures and functional partitioning scheme can be utilized and adopted for the future CABAC decoder design.

Future research directions of CABAC encoder design can be: (1) Throughput enhancement of context-dependent SE coding with multiple bin per cycle coding throughput, which is more difficult than residual SE acceleration; (2) Acceleration of the current RDO coding scheme by reducing pipeline filling and empty delay of each RDO mode; (3) Parallel CABAC coding using multiple independent processing units of ASIC cores or general multiple core processors, which is a tradeoff between coding acceleration and compression efficiency, and it is not compatible to the current H.264 standard.

Bibliography

Standards and Technical Specifications

[AMBA] AMBA^{TM} (1999) Specifications (Rev 2.0), Vol. ARM

[ISO/IEC 10918-1] ISO/IEC JTC1 (1991) Digital compression and coding of continuous tone still images. Part 1, requirements and guidelines. Draft international standard 10918, Nov

[ISO/IEC 11172] ISO/IEC 11172 (1991) Coding of moving pictures and associated audio – for digital storage media at up to about 1.5 Mb/s

[ISO/IEC 13818-2] ISO/IEC 13818-2 (1993) Generic coding of moving pictures and associated audio. Video, Nov

[ITU-T Rec. H.261] ITU-T Rec. H.261 (1993) Video codec for audio-visual services at p $\times$ 64 kbit/s

[ITU-T Rec. H.263] ITU-T Rec. H.263 (1996) Video codec for low-bit rate communications

[ISO/IEC 14496-2] Information technology – coding of audio-visual objects—Part 2: visual. ISO/IEC JTC1, ISO/IEC International Standard 14496-2 (MPEG-4 Visual Version 1)

[ISO/IEC 14496-10] Advanced video coding for generic audiovisual services. ITU-T and ISO/IEC, ITU-T recommendation H.264 and ISO/IEC international standard 14496, Part 10 (AVC), 2003

[JBIG] Information technology – coded representation of picture and audio information – progressive bi-level image compression. ISO/IEC and ITU-T, ISO/IEC International Standard 11544 and ITU-T Recommendation T.82, 1993

[JPEG] Information technology-JPEG-digital compression and coding of continuous-cone still image-Part 1: requirement and guidelines. ISO/IEC and ITU-T, ISO/IEC International Standard 10918-1 and ITU-T Recommendation T.81, 1994

[JPEG2000] Information technology—JPEG2000 image coding system—Part 1: core coding system. ISO/IEC, ISO/IEC International Standard 15444-1, 2000

[Wishbone] WISHBONE system-on-a-chip interconnection architecture for portable IP cores, revision B.3 specification, Vol. OPENCORES, 2002

Websites

[WSAstro] Astro, Synopsys. http://www.synopsys.com

[WSCACTI] HP lab, RAM and cache modeling tool. http://quid.hpl.hp.com:9081/cacti/sram.y

[WSCS] ChipScope Pro, Xilinx. http://www.xilinx.com/ise/optional_prod/cspro.htm

[WSDC] Design Compiler, Synopsys, http://www.synopsys.com

[WSh263] Sources for H.263. ftp://bonde.nta.no: /pub/tmn/software

[WSISE] ISE Webpack Software, Xilinx. http://www.xilinx.com/ise/logic_design_prod/webpack.htm

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6,

[WSJM] Sources for H.264/AVC JVT reference codec software: JM 12.4. http://iphome.hhi.de/suehring/tml/
[WSjpeg] Sources for JPEG. ftp://ftp.uu.net: /graphics/jpeg/
[WSMB] MicroBlaze Processor, Xilinx. http://www.xilinx.com/products/design_resources/proc_central/microblaze.htm
[WSMBIST] MBISTArchitect, Mentor Graphics. http://www.mentor.com/products/silicon-yield/memorytest/mbistarchitect/
[WSModelsim] ModelSim SE 6.1b, mentor graphics. http://www.model.com/downloads/default.asp
[WSmpeg] Sources for MPEG, ISO/IEC. http://www.mpeg.org
[WSPC] Power Compiler, Synopsys. http://www.synopsys.com
[WSPIN] PIN software analysis tool, University of Colorado. http://rogue.colorado.edu/pin/
[WStree96] Sources for tree-search VQ. ftp://isdl.ee.washington.edu: /pub/VQ/code/

Books, Book Chapters, and Papers

[Abramson63] Abramson N (1963) Information theory and coding. McGraw-Hill Book Co., Inc., New York, NY
[Ahmed74] Ahmed N, Natarajan T, Rao KR (Jan 1974) Discrete cosine transform. IEEE Trans Comput C-23(1): 90–93
[Agostini06] Agostini L, Porto R, Guntzel J, Saraiva Silva I, Bampi S, (2006) High throughput multitransform and multiparallelism IP for H.264/AVC video compression standard. In: Proceedings of IEEE international symposium on circuits and systems, Island of Kos, Greece, p 4
[Alle06] Alle M, Biswas J, Nandy SK (2006) High performance VLSI architecture design for H.264 CAVLC decoder. In: Proceedings of international conference on application-specific systems, architectures and processors, Steamboat Springs, Colorado, USA, pp 317–322
[Ba06] Ba S-N, Altunbasak Y, Ates H (2006) Low complexity inter-mode selection for H.264. In: Proceedings of IEEE international conference on image processing, Atlanta, Georgia, USA, pp 1349–1352
[Babionitakis06] Babionitakis K, Lentaris G, Nakos K, Reisis D, Vlassopoulos N, Doumenis G, Georgakarakos G, Sifnaios J (2006) An efficient H.264 VLSI advanced video encoder. In: Proceedings of 13th IEEE international conference on electronics, circuits and systems, Nice, France, pp 545–548
[Baik07] Baik H, Sihn K-H, Kim Y-i, Bae S, Han N, Song HJ (2007) Analysis and parallelization of H.264 decoder on cell broadband engine architecture. In: Proceedings of IEEE international symposium on signal processing and information technology, Cairo, Egypt, pp 791–795
[Bjøntegaard02] Bjøntegaard G, Lillevold K (2002) Context-adaptive VLC (CAVLC) coding of coefficients. 3rd meeting of joint video team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Doc. JVT-C028, Fairfax, Virginia, USA
[Chang05] Chang H, Lin C, Guo J (2005) A novel low-cost high-performance VLSI architecture for MPEG-4 AVC/H.264 CAVLC decoding. In: Proceedings of international symposium on circuits and systems, Kobe, Japan, pp 6110–6113
[Chang07a] Chang C-H, Chen J-W, Chang H-C, Yang Y-C, Wang J-S, Guo J-I (2007) A quality scalable H.264/AVC baseline intra encoder for high definition video applications. In: Proceedings of IEEE workshop on signal processing systems, Shanghai, China, pp 521–526
[Chang07b] Chang SC, Cheng C-C, Chen L-G (2007) System architecture design methodology for H.264/AVC encoder. In: Proceedings of IEEE international symposium on consumer electronics, Dallas, Texas, USA, pp 1–5
[Chao06] Chao Y, Wei S, Yang J, Liu B (2006) Combined CAVLC decoder and inverse quantizer for efficient H.264/AVC decoding. In: Proceedings of Asia and Pacific conference on circuits and systems, Singapore, pp 259–262

[Chen77] Chen WH, Smith CH, Fralick SC (Sep 1977) A fast computational algorithm for the discrete cosine transform. IEEE Trans Commun COM-25(9): 1004–1009

[Chen05] Chen J-W, Chang C-R, Lin Y-L (2005) A hardware accelerator for context-based adaptive binary arithmetic decoding in H.264/AVC. Proc IEEE Int Symp Circ Syst 5:4525–4528, Kobe, Japan

[Chen06a] Chen T-C, Chien S-Y, Huang Y-W, Tsai C-H, Chen C-Y, Chen T-W, Chen L-G (2006) Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder. IEEE Trans Circ Syst Video Technol 16(6):673–688

[Chen06b] Chen T-C, Lian C-J, Chen L-G (2006) Hardware architecture design of an H.264/AVC video codec. In: Proceedings of Asia and South Pacific conference on design automation, Yokohama, Japan

[Chen06c] Chen T, Huang Y, Tsai C, Hsieh B, Chen L (2006) Architecture design of context-based adaptive variable-length coding for H.264/AVC. IEEE Trans Circuits Syst II 53(9): 832–836

[Chen06d] Chen Y-J, Tsai C-H, Chen L-G (2006) Architecture design of area-efficient SRAM-based multi-symbol arithmetic encoder in H.264/AVC. In: Proceedings of IEEE international symposium on circuits and systems, Island of Kos, Greece, pp 2621–2624

[Chen07a] Chen J-W, Lin Y-L (2007a) A high-performance hardwired CABAC decoder. In: Proceedings of IEEE international conference on acoustics, speech and signal processing, Honolulu, Hawaii, USA, pp II-37–II-40

[Chen07b] Chen J-L, Lin Y-K, Chang T-S (2007b) A low cost context adaptive arithmetic coder for H.264/MPEG-4 AVC video coding. In: Proceedings of IEEE international conference on acoustics, speech and signal processing, Honolulu, Hawaii, USA, pp II-105–II-108

[Chien06] Chien C-D, Lu K-P, Shih Y-H, Guo J-I (2006) A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications. In: Proceedings of Asia and Pacific conference on circuits and systems, Singapore, pp 3837–3841

[Cho91] Cho NI, Lee SU (Mar 1991) Fast algorithm and implementation of 2-D DCT. IEEE Trans Circuits Syst 38(3):297–305

[Chou89] Chou CH, Chen YC (Oct 1989) A VLSI architecture for real-time and flexible image template matching. IEEE Trans Circuits Syst 36(10):1336–1342

[Eeckhaut06] Eeckhaut H, Christiaens M, Stroobandt D, Nollet V (2006) Optimizing the critical loop in the H.264/AVC CABAC decoder. In: Proceedings of IEEE international conference on field programmable technology, Bangkok, Thailand, pp 113–118

[Feig92] Feig E, Winograd S (Sep 1992) Fast algorithms for the discrete cosine transform. IEEE Trans Signal Process 40(9):2174–2193

[Feng95] Feng J, Lo K-T, Mehrpour H, Karbowiak AE (Oct 1995) Adaptive block matching motion estimation algorithm using bit-plane matching. Proc IEEE Int Conf Image Process 3:496–499, Washington, DC, USA

[Flordal06] Flordal O, Wu D, Liu D (2006) Accelerating CABAC encoding for multi-standard media with configurability. In: Proceedings of 20th international parallel and distributed processing symposium, Anchorage, Alaska, USA

[Gharavi90] Gharavi H, Mills M (May 1990) Block-matching motion estimation. IEEE Trans Circuits Syst 37(5):649–651

[Golomb66] Golomb S (1966) Run-length encodings. IEEE Trans Inf Theory 12(3):399–401

[Gordon04] Gordon S, Marple D, Wiegand T (2004) Simplified use of 8×8 transforms – updated proposal and results. Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, 11th meeting, JVT-K028, Munich, Germany, Mar

[Golwelkar07] Golwelkar A, Woods JW (2007) Motion-compensated temporal filtering and motion vector coding using biorthogonal filters. IEEE Trans Circuits Syst Video Technol 17(4):417–428

[Ha05] Ha VHS, Shim W-S, Kim J-W (2005) Real-time MPEG-4 AVC/H.264 CABAC entropy coder. In: Proceedings of international conference on consumer electronics, Las Vegas, Nevada, USA, pp 255–256

[Habibi77] Habibi A (Nov 1977) Survey of adaptive image coding techniques. IEEE Trans Commun COM-25(11):1275–1284

[Heising01] Heising G, Marpe D, Cycon HL, Petukhov AP (2001) Wavelet-based very low bit-rate video coding using image warping and overlapped block motion compensation. IEE Proc Vis Image Signal Process 148(2):93–101

[Ho06] Ho BL (2006) Performance and complexity analyses of H.264/AVC CABAC entropy coder. Master of engineering thesis, Department of Electrical and Computer Engineering, National University of Singapore

[Hu08] Hu H, Sun J, Xu J (2008) High performance architecture design of CAVLC encoder in H.264/AVC. In: Proceedings of congress on image and signal processing, Sanya, Hainan, China, pp 613–616

[Huang05] Huang Y-W, Chen T-C, Tsai C-H, Chen C-Y, Chen T-W, Chen C-S, Shen C-F, Ma S-Y, Wang T-C, Hsieh B-Y, Fang H-C, Chen L-G (2005) A 1.3TOPS H.264/AVC single-chip encoder for HDTV applications. Proceedings of IEEE Int Solid State Circuits Conf 1:128–588 San Francisco, California, USA

[Huang08] Huang T-Y, Jian G-A, Chu J-C, Su C-L, Guo J-I (2008) Joint algorithm/code-level optimization of H.264 video decoder for mobile multimedia applications. In Proceedings of IEEE international conference on acoustics, speech and signal processing, Las Vegas, Nevada, USA, pp 2189–2192

[Huffman52] Huffman DA (1952) A method for the construction of minimum redundancy codes. Proc IRE 40(10):1098–1101

[Inata08] Inata K, Sasamoto M, Nonaka T, Komi H (2008) System architecture of H.264/AVC codec LSI for digital HD camcorder. In: Proceedings of international conference on consumer electronics, Las Vegas, Nevada, USA, pp 1–2

[Jain81] Jain JR, Jain AK (Dec 1981) Displacement measurement and its application in inter-frame image coding. IEEE Trans Commun COM-29(12):1799–1808

[Jamaa06] Jamaa SB, Kieffer M, Duhamel P (2006) Controlled complexity map decoding of CABAC encoded data. In: Proceedings of IEEE international conference on multimedia and expo, Toronto, Canada, pp 1441–1444

[Kant06] Kant S, Mithun U, Gupta PSSBK (2006) Real time H.264 video encoder implementation on a programmable DSP processor for videophone applications. In: Proceedings of international conference on consumer electronics, Las Vegas, Nevada, USA, pp 93–94

[Kim95] Kim Y, Rim CS, Min B (1995) A block matching algorithm with 16:1 subsampling and its hardware design. ISCAS'95, Seattle, Washington, USA, pp 613–616

[Kim04] Kim H, Altunhasak Y (2004) Low-complexity macroblock mode selection for H.264-AVC encoders. In: Proc Int Conf Image Process 2:765–768, Singapore

[Kim06a] Kim D, Jung E, Park H, Shin H, Har D (2006) Implementation of high performance CAVLC for H.264/AVC video codec. In: Proceedings of 6th international workshop on SOC for real-time applications, Cairo, Egypt, pp 20–23

[Kim06b] Kim C-H, Park I-C (2006) High speed decoding of context-based adaptive binary arithmetic codes using most probable symbol prediction. In: Proceedings of IEEE international symposium on circuits and systems, Island of Kos, Greece

[Koga81] Koga T, Iinuma K, Hirano A, Iijima Y, Ishiguro T (1981) Motion compensated inter-frame coding for video conferencing. Proceedings of the national telecommunication conference, pp G5.3.1–5.3.5, Nov

[Kordasiewicz07] Kordasiewicz RC, Gallant MD, Shirani S (2007) Encoding of affine motion vectors. IEEE Transactions on Multimedia, 9(7):1346–1356

[Kuo06] Kuo C-C, Lei S-F (2006) Design of a low power architecture for CABAC encoder in H.264. In: Proceedings of IEEE Asia Pacific conference on circuits and systems, Singapore, pp 243–246

[Lahti05] Lahti J, Juntunen JK, Lehtoranta O, Hamalainen TD (2005) Algorithmic optimization of H.264/AVC encoder. Proceedings of IEEE Int Symp Circuits Syst 4:3463–3466, Kobe, Japan

[Langdon84] Langdon GG (1984) An introduction to arithmetic coding. IBM J Res Dev 28: 135–149

[Le06] Le TM, Tian XH, Ho BL, Nankoo J, Lian Y (2006) System-on-chip design methodology for a statistical coder. In: Proceedings of 7th IEEE international workshop on rapid system prototyping, Porto Alegre, Brazil, pp 82–90

[Lee84] Lee BG (Dec 1984) A new algorithm to compute discrete cosine transform. IEEE Trans Acoust Speech Signal Process ASSP-32(6):1243–1245

[Lee08] Lee S, Park S, Han J, Eum N, Jongwon P (2008) A 40MHZ dedicated hardware H.264/AVC video encoder with the reducing memory access scheme. In: Proceedings of IEEE international symposium on consumer electronics, Algarve, Portugal, pp 1–4

[Levine07] Levine D, Lynch WE, Tho L-N (2007) Observations on error detection in H.264. In: Proceedings of 50th Midwest symposium on circuits and systems, Montreal, Canada, pp 815–818

[Li06a] Li L, Song Y, Ikenaga T, Goto S (2006) A CABAC encoding core with dynamic pipeline for H.264/AVC main profile. In: Proceedings of IEEE Asia Pacific conference on circuits and systems, Singapore, pp 760–763

[Li06b] Li M, Wu W (2006) A high throughput binary arithmetic coding engine for H.264/AVC. In: Proceedings of 8th international conference on solid-state and integrated circuit technology, Shanghai, China, pp 1914–1918

[Li06c] Li Y, Xiong H, Song L, Yu S (2006) A context-based error detection strategy into H.264/AVC CABAC. In: Proceedings of IEEE international conference on multimedia and expo, Toronto, Canada, pp 689–692

[Li07a] Li B, Zhang D, Fang J, Wang L, Zhang M (2007) A high-performance VLSI architecture for CABAC decoding in H.264/AVC. In Proceedings of 7th international conference on ASIC, Guilin, China, pp 790–793

[Li07b] Li Y, Qu Y, He Y (2007) Memory cache based motion compensation architecture for HDTV H.264/AVC decoder. In Proceedings of IEEE international symposium on circuits and systems, New orleans, Louisiana, USA, pp 2906–2909

[Lin06a] Lin H-C, Wang Y-J, Cheng K-T, Yeh S-Y, Chen W-N, Tsai C-Y, Chang T-S, Hang H-M (2006) Algorithms and DSP implementation of H.264/AVC. In: Proceedings of Asia and South Pacific conference on design automation Yokohama, Japan

[Lin06b] Lin JH, Parhi KK (2006) Parallelization of context-based adaptive binary arithmetic coders. IEEE Trans Signal Process 54(10):3702–3711

[Lin08a] Lin H, Lu Y, Liu B, Yang J (2008) A highly efficient VLSI architecture for H.264/AVC CAVLC decoder. IEEE Trans Multimedia, 10(1):31–42

[Lin08b] Lin Y-K, De-Wei L, Chia-Chun L, Tzu-Yun K, Sian-Jin W, Wei-Cheng T, Wei-Cheng C, Tian-Sheuan C (2008) A 242mW 10mm2 1080p H.264/AVC high-profile encoder chip. In: Proceedings of IEEE international solid-state circuits conference, San Francisco, California, USA, pp 314–615

[Lin08c] Lin Y-K, De-Wei L, Chia-Chun L, Tzu-Yun K, Sian-Jin W, Wei-Cheng T, Wei-Cheng C, Tian-Sheuan C (2008) A 242mW, 10mm2 1080p H.264/AVC high profile encoder chip. In: Proceedings of 45th ACM/IEEE design automation conference, Anaheim, California, USA, pp 78–83

[Liu93] Liu B, Zaccarin A (Apr 1993) New fast algorithms for the estimation of block motion vectors. IEEE Trans Circuits Syst Video Technol 3(2):148–157

[Liu07a] Liu P-S, Chen J-W, Lin Y-L (2007a) A hardwired context-based adaptive binary arithmetic encoder for H. 264 advanced video coding. In Proceedings of international symposium on VLSI design, automation and test, Hsin Chu, Taiwan, China, pp 1–4

[Liu07b] Liu Z, Yang S, Ming S, Shen L, Lingfeng L, Ishiwata S, Nakagawa M, Goto S, Ikenaga T (2007) A 1.41 W H.264/AVC real-time encoder SOC for HDTV1080P. In Proceedings of IEEE symposium on VLSI circuits, Kyoto, Japan, pp 12–13

[Lo07] Lo C-C, Zeng Y-J, Shieh M-D (2007) Design and test of a high-throughput CABAC encoder. In: Proceedings of IEEE region 10 conference, Taipei, Taiwan, China, pp 1–4

[Lu97] Lu J, Liou ML (Apr 1997) A simple and efficient search algorithm for block-matching motion estimation. IEEE Trans Circuits Syst Video Technol 7(2)

[Ma05] Ma S, Gao W, Lu Y (2005) Rate-distortion analysis for H.264/AVC video coding and its application to rate control. IEEE Trans Circuits Syst Video Technol 15(12):1533–1544

[Mallat89] Mallat SG (Jul 1989) A theory for multi-resolution signal representation: the wavelet decomposition. IEEE Trans Pattern Anal Mach Intell 11(7):674–693

[Malva03] Malvar HS, Hallapuro A, Karczewicz M, Kerofsky L (Jul 2003) Low-complexity transform and quantization in H.264/AVC. IEEE Trans Circuits Syst Video Technol 13(7):598–603

[Marpe97] Marpe D, Cycon HL (1997) Efficient pre-coding techniques for wavelet-based image compression. In: Proceedings of picture coding symposium, Berlin, Germany, pp 45–50

[Marpe99] Marpe D, Cycon HL (1999) Very low bit-rate video coding using wavelet-based techniques. IEEE Trans Circuits Syst Video Technol 9(4):85–94

[Marpe01a] Marpe D, Blättermann G, Heising G, Wiegand T (2001a) Further results for CABAC entropy coding scheme. ITU-T SG16/Q.6 Doc. VCEG-M59, Austin, TX, USA

[Marpe01b] Marpe D, Blättermann G, Wiegand T (2001b) Adaptive codes for H.26L. ITU-T SG16/Q.6 Doc. VCEG-L13, Eibsee, Germany

[Marpe01c] Marpe D, Blättermann G, Wiegand T (2001c) Improved CABAC. ITU-T SG16/Q.6 Doc. VCEG-O18, Pattaya, Thailand

[Marpe03a] Marpe D, Schwarz H, Wiegand T (2003a) Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. IEEE Trans Circuits Syst Video Technol 13(7):620–636

[Marpe03b] Marpe D, Wiegand T (2003b) A highly efficient multiplication-free binary arithmetic coder and its application in video coding. In: Proceedings of Int Conf Image Process 3:II-263–266 Barcelona, Spain

[Mitchell93] Mitchell J, Pennebaker W (1993) JPEG: still image data compression standard. Van Nostrand Reinhold, New York, NY

[Moffat02] Moffat A, Turpin A (2002) Compression and coding algorithms. Kluwer Academic Publishers, Dordrecht

[Moon05] Moon Y, Kim G, Kim J (2005) An efficient decoding of CAVLC in H.264/AVC video coding standard. IEEE Trans Consum Electron 51(3):933–938

[Mrak03] Mrak M, Marpe D, Wiegand T (2003) A context modeling algorithm and its application in video compression. Proc Int Conf Image Process, 2:III-845–848 Barcelona, Spain

[Muller05] Muller K, Smolic A, Kautzner M, Eisert P, Wiegand T (2005) Predictive compression of dynamic 3D meshes. In: Proceedings of IEEE international conference on image processing, Genoa, Italy, pp I-621–624

[Murachi08] Murachi Y, Mizuno K, Miyakoshi J, Hamamoto M, Iinuma T, Ishihara T, Fang Y, Jangchung L, Kamino T, Kawaguchi H, Yoshimoto M (2008) A sub 100 mW H.264/AVC MP@L4.1 integer-pel motion estimation processor VLSI for MBAFF encoding. In: Proceedings of IEEE international symposium on circuits and systems, Seattle, Washington, USA, pp 848–851

[Nararajan97] Natarajan B, Bhaskaran V, Konstatinides K (1997) Low-complexity block-based ME via one-bit transform. IEEE Trans Circuits Syst Video Technol 7(4):702–706

[Netravali80] Netravali AN, Limb JO (Mar 1980) Transform picture coding. Proc IEEE 68(3): 366–407

[Netravali89] Netravali AN, Haskell BG (1989) Digital pictures: representation and compression. Plenum Press, New York, NY

[Nieto06] Nieto M, Salgado L, Cabrera J (2006) Fast mode decision on H.264/AVC main profile encoding based on PSNR predictions. In: Proceedings of IEEE international conference on image processing, Atlanta, Georgia, USA, pp 49–52

[Nunez06a] Nunez-Yanez JL, Chouliaras VA, Alfonso D (2006a) Hardware assisted rate distortion optimization with embedded CABAC accelerator for the H.264 advanced video codec. In: Proceedings of international conference on consumer electronics, Las Vegas, Nevada, USA, pp 95–96

[Nunez06b] Nunez-Yanez JL, Chouliaras VA, Alfonso D, Rovati FS (2006b) Hardware assisted rate distortion optimization with embedded CABAC accelerator for the H.264 advanced video codec. IEEE Trans Consum Electron 52(2):590–597

[Osorio04] Osorio RR, Bruguera JD (2004) Arithmetic coding architecture for H.264/AVC CABAC compression system. In: Proceedings of Euromicro symposium on digital system design, Rennes, France, pp 62–69

[Osorio05] Osorio RR, Bruguera JD (2005) A new architecture for fast arithmetic coding in H.264 advanced video coder. In: Proceedings of 8th Euromicro conference on digital system design, Lugano, Switzerland, pp 298–305

[Osorio06] Osorio RR, Bruguera JD (2006) High-throughput architecture for H.264/AVC CABAC compression system. IEEE Trans Circuits Syst Video Technol 16(11):1376–1384

[Pan04] Pan F, Rahardja LS, Lim KP, Wu LD, Wu WS, Zhu C, Ye W, Liang Z (2004) Fast intra mode decision algorithm for H.264-AVC video coding. Proc Int Conf Image Process 2:781–784, Singapore

[Pasco76] Pasco RC (1976) Source coding algorithms for fast data compression, PhD thesis, Department of Electrical Engineering, Stanford University

[Pastuszak05] Pastuszak G (2005) A high-performance architecture for embedded block coding in JPEG2000. IEEE Trans Circuits Syst Video Technol 15(9):1182–1191

[Pennebaker88] Pennebaker WB, Mitchell JL, Langdon JGG, Arps RB (1988) An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. IBM J Res Dev 32(6):717–726

[Rabaey06] Rabaey JM (2006) Issues in low power design – minimizing active power http://citeseerx.ist.psu.edu/viewdoc/download? doi=10.1.1.92.7505&rep=rep1&type=pdf

[Rahman07] Rahman CA, Badawy W (2007) CAVLC encoder design for real-time mobile video applications. IEEE Trans Circuits Syst II 54(10):873–877

[Rao90] Rao KR, Yip P (1990) Discrete cosine transform: algorithms, advantages, and applications. Academic Press, Boston, MA

[Richardson03] Richardson IEG (2003) H.264 and MPEG-4 video compression: video coding for next-generation multimedia. Wiley, Chichester

[Rissanen76] Rissanen JJ (1976) Generalized kraft inequality and arithmetic coding. IBM J Res Dev 20(198):198–203

[Roese77] Roese JA, Pratt WK, Robinson GS (1977) Interframe cosine transform image coding. IEEE Trans Commun COM-25(11):1329–1338

[Sanchez08] Sanchez V, Nasiopoulos P, Abugharbieh R (2008) Efficient 4D motion compensated lossless compression of dynamic volumetric medical image data. In: Proceedings of IEEE international conference on acoustics, speech and signal processing, pp 549–552 Las Vegas, Nevada, USA

[Sayed06] Sayed M, Amer I, Badawy W (2006) Towards an H.264/AVC full encoder on chip: an efficient real-time VBSME ASIC chip. In: Proceedings of IEEE international symposium on circuits and systems Island of Kos, Greece

[Schwarz07] Schwarz H, Wiegand T (2007) R-D optimized multi-layer encoder control for SVC. In: Proceedings of IEEE international conference on image processing, San Antonio, Texas, USA, pp II-281–284

[Sehoon07] Sehoon Y, Vetro A (2007) RD-optimized view synthesis prediction for multiview video coding. In: Proceedings of IEEE international conference on image processing, San Antonio, Texas, USA, pp I-209–212

[Shannon48] Shannon CE (1948) A mathematical theory of communication. Bell Syst Tech J 27:379–423, 623–656

[Shi08] Shi B, Zheng W, Lee H-S, Li D-X, Zhang M (2008) Pipelined architecture design of H.264/AVC CABAC real-time decoding. In: Proceedings of 4th IEEE international conference on circuits and systems for communications, Shanghai, China, pp 492–496

[Shojania05] Shojania H, Sudharsanan S (2005) A high performance CABAC encoder. In: Proceedings of 3rd international IEEE-NEWCAS conference, Montreal, Canada, pp 315–318

[Slattery98] Slattery MJ, Mitchell JL (1998) The Qx-coder. IBM J Res Dev 42(6):767–784

[Son08] Son W, Park I-C (2008) Prediction-based real-time CABAC decoder for high definition H.264/AVC. In: Proceedings of IEEE international symposium on circuits and systems, Seattle, Washington, USA, pp 33–36

[Srinivasan85] Srinivasan R, Rao KR (Aug 1985) Predictive coding based on efficient motion estimation. IEEE Trans Commun COM-33:888–896

[Sudharsanan05] Sudharsanan S, Cohen A (2005) A hardware architecture for a context-adaptive binary arithmetic coder. In: Proceedings of SPIE embedded processors for multimedia and communications II, San Jose, USA, pp 104–112

[Sullivan98] Sullivan GJ, Wiegand T (1998) Rate-distortion optimization for video compression. IEEE Signal Process Mag 15(6):74–90

[Sun07] Sun C, Wang H-J, Li H, Kim T-H, Yu X-B (2007) An efficient context modeling algorithm for motion vectors in CABAC. In: Proceedings of IEEE international symposium on signal processing and information technology, Cairo, Egypt, pp 796–800

[Taubman00] Taubman DS (2000) High performance scalable image compression with EBCOT. IEEE Trans Image Process 9(7):1158–1170

[Taubman02] Taubman DS, Marcellin MW (2002) JPEG2000 image compression fundamentals, standards and practice. Kluwer Academic Publishers, Dordrecht

[Teuhola78] Teuhola J (1978) A compression method for clustered bit-vectors. Inf Process Lett 7(10):308–311

[Tian08] Tian XH, Le TM, Jiang X, Lian Y (2008) A HW CABAC encoder with efficient context access scheme for H.264/AVC. In: Proceedings of IEEE international symposium on circuits and systems, pp 37–40 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4534149

[Tian09] Tian XH, Le TM, Jiang X, Lian Y (Sep 2009) Full RDO-support power-aware CABAC encoder with efficient context access. IEEE Trans Circuits Syst Video Technol 19(9):1262–1273

[Tsa07] Tsa T, Fang D, Pan Y (2007) A hybrid CAVLD architecture design with low complexity and low power considerations. In: Proceedings of IEEE international conference on multimedia and expo, pp 1910–1913 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4284552

[Tsai05a] Tsai C-H, Huang Y-W, Chen L-G (2005a) Algorithm and architecture optimization for full-mode encoding of H.264/AVC intra prediction. Proceedings of 48th Midwest symposium on circuits and systems, vol 1, pp 47–50 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=10622

[Tsai05b] Tsai C-Y, Chen T-C, Chen T-W, Chen L-G (2005b) Bandwidth optimized motion compensation hardware design for H.264/AVC HDTV decoder. In: Proceedings of 48th Midwest symposium on circuits and systems, vol 2, pp 1199–1202, 2005b http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=10622

[Tsai06] Tsai C-Y, Chen T-C, Chen L-G (2006) Low power entropy coding hardware design for H.264/AVC baseline profile encoder. In Proceedings of IEEE international conference on multimedia and expo, pp 1941–1944 http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=4036510

[Wang05] Wang Y, Yu S (2005) Joint source-channel decoding for H.264 coded video stream. IEEE Trans Consum Electron 51(4):1273–1276

[Weber02] Weber M (2001) Arbiters: design ideas and coding styles. In: Proceedings of SNUG of synopsys, Boston, MA http://www.synopsys.com/community/snug/pages/proceedingLp.aspx?loc=Boston&locy=2001

[Wei07] Wei Z, Tang KL, Ngan KN (2007) Implementation of H.264 on mobile device. IEEE Trans Consum Electron 53(3):1109–1116

[Wiegand03] Wiegand T, Sullivan GJ, Bjontegaard G, Luthra A (2003) Overview of the H.264/AVC video coding standard. IEEE Trans Circuits Syst Video Technol 13(7):560–576

[Witten87] Witten IH, Neal RM, Cleary JG, (1987) Arithmetic coding for data compression. Commun ACM 30(6):520–540

[Wu07] Wu Y, Woods JW (2007) Scalable motion vector coding based on CABAC for MC-EZBC. IEEE Trans Circuits Syst Video Technol 17(6):790–795

[Yi07] Yi Y, Park IC (2007) High-speed H.264/AVC CABAC decoding. IEEE Trans Circuits Syst Video Technol 17(4):490–494

[Yu06] Yu L, Li J, Shen Y (2006) Fast frame/field coding for H.264/AVC. In: Proceedings of international conference on digital telecommunications, pp 18–18 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=11153

[Zhang07] Zhang L, Wu X, Zhang N, Gao W, Wang Q, Zhao D (2007) Context-based arithmetic coding reexamined for DCT video compression. In Proceedings of IEEE international symposium on circuits and systems, pp 3147–3150 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4252534

Index