SYSTEM-ON-CHIP DESIGN OF A HIGH PERFORMANCE LOW POWER FULL HARDWARE CABAC ENCODER IN H.264/AVC

TIAN XIAOHUA

(M.Eng, HUST)

A THESIS SUBMITTED FOR THE DEGREE OF PH.D.

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2009

Acknowledgements

First of all, I would like to thank my supervisors Dr. Le M. Thinh and Prof. Lian Yong for their advices, encouragement, and long-term supports during my Ph. D. study and research work. Without these two great mentors, I would not complete my research work successfully.

Thanks to the colleagues of our research group including Mr. Jiang Xi, Ho Boon Leng, Shyam Krishnamurthy, Hong Zhiqian, Thu Trang, Esmond Teo Haochun, and John Nankoo for their supports, suggestions, and helpful discussions. Without them, I could not build up the complete scheme of this CABAC encoder design of my thesis.

Thanks to my friends in VLSI lab including Wei Ying, Zhang Wenjuan, Zhu Youpan, Chen Xiaolei, Zhang Xiaoyang, Bai Na, Zhang Jinghua, Yang Zhenglin, Pu Yu, Zou Xiaodan, Xiaoyuan, Wu Liqun, Yu Heng, Li Yanhui, San Jeow, Cheng Xiang, Tan Jun, Chang Xiaofei, Niu Tianfang, Wang Lei, Qiu Lin, Raja, Amit, Lynn, John, Shakith, my seniors Yu Jianghong, Yu Rui, Chen Jianzhong, He Lin, Hu Yingping, Tong Yan, Cen Lin, Gu Jun, and many others.

Thanks for the valuable advices and help from Mr. Jiang Xiping, Dr. Ha Yajun, Prof. Xu Yong Ping, Ms. Zheng Huanqun, Mr. Teo Seow Miang, Prof. Zhu Minghua, *et al*. for my research work.

Finally, I would like to thank my dear Father and Mother, my Grandma, uncles and aunts, Wenxiu, Liu Yu, Tian Jun, Xiang Li, Tian Zhenzhen, Li Jie, Li Chi, Fang Congbiao, my friends Wang Enbo, Zhou Jinxin, Liu Chunhui, Zhang Jing, Teng Mingqing, Wen Qiang, Liang Kun, *et al*. for their encouragements that support me to complete this thesis.

Abstract

Context-based Adaptive Binary Arithmetic Coding (CABAC) is the entropy coding tool adopted in Main and High profiles of H.264/AVC video coding standard. CABAC provides significantly higher compression ratio than Baseline profile entropy coder CAVLC. Rate-Distortion Optimization (RDO) is another important technique that improves the encoding performance of H.264/AVC. It is necessary to support both CABAC and RDO in the high quality and high definition H.264/AVC applications; however, this results in significantly increased computational complexity. Due to the sequential 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. This work focuses on high performance circuit design of CABAC encoder IP targeting at Main Profile of H.264/AVC.

SoC-based design flow is explored during the CABAC encoder IP design, including steps of encoder performance and complexity analysis; system specification; HW/SW partitioning that minimizes computation complexity on the host processor and data transfer on system bus; HW functional partitioning that maximizes encoding parallelism; HW function block design; SoC feature insertion including system bus interface and interconnection IP design; circuit implementation and verification, etc. The encoder is designed and fully verified at RTL level, gate level, and post-layout stage targeting at 0.13um CMOS process. FPGA prototyping is also completed successfully.

In order to accelerate sequential and highly data dependent procedure of CABAC and optimize circuit performance, various design methodologies are explored in this work, including: prefetch and local buffering for frequent accessed data to reduce data fetch delay; precalculation to reduce critical path length; pipeline implementation of complex sequential computation steps to achieve higher clock frequency; SRAM access optimization with context line access & buffering and context RAM reallocation to significantly reduce RAM access frequency and dynamic power; parallel processing of function blocks of different throughput with FIFO insertion; system power reduction with clock gating insertion, etc.

This work provides the only reported CABAC encoder design that achieves high processing speed of real time coding in CIF format full RDO mode and in HDTV 720p format RDO-off mode. The compression efficiency of the proposed encoder is the best compared to the reported designs, because of solving design difficulty of CABAC coding in RDO mode. Encoder power consumption is the lowest, consuming only 0.79 mW at HDTV 720p60 8.9 Mbps RDO-off mode coding. Only this work provides complete SoC-based IP solution of CABAC encoder that can efficiently support different H.264 coding configurations including RDO-off, fast RDO, and full RDO mode, and the application range of the IP is wider, from real time coding to high quality compression. This work enhances performance of both CABAC encoder and H.264 video coding system and achieves global performance optimization, with utilization of encoder design flexibility.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

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 spatial, temporal, and code word domain. Several video coding standards have been established since 1980's to specify video coding techniques utilized for different applications, including H.261 [1], MPEG-1 [2], MPEG-2 [3], H.263 [4], MPEG-4 Part 2 [5], and H.264/AVC [6]. H.261 is the first video coding standard targeting at low delay, slow motion applications such as video conference. MPEG-1 introduces half-pixel motion estimation and bi-direction motion estimation (ME), with perceptual-based quantization, similar to JPEG [7]. MPEG-2 (also known as H.262) supports interlaced video format and broadcasting quality video coding. H.263 achieves a significant improvement of video compression especially at low bit rate, with more efficient ME and techniques of variable block size ME and arithmetic coding adopted in H.263 Annex. MPEG-4 Part 2 adopts ¼-pixel ME, and several commercial codecs are designed based on Advanced Simple Profile (ASP) of the standard. The latest video coding standard H.264/AVC (MPEG-4 Part 10) [6] is developed to target at a wide range of applications and high compression capability.

1.1 Overview of H.264/AVC Standard

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% bit-rate 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 at 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. 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 frame or field is further partitioned into one or more *slices*, and each slice consists of a sequence of MBs. Slice is the smallest self-contained [8] 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). Block-based hybrid video coding approach is utilized in VCL layer.

The block diagrams of MB encoding and decoding of VCL layer are shown in Figure 1-1. As shown in Figure 1-1(a), MBs in each slice are sequentially processed at the encoder. Intra prediction is applied to reduce spatial redundancy of coding MB by predicting pixels of current MB based the boundary pixels of neighboring coded MBs. As only prediction residual values of intra-coded MBs are encoded, compression efficiency is enhanced. Inter prediction includes ME and motion compensation (MC), which are applied to inter-coded MBs to reduce temporal redundancy. Precise motion estimation is achieved through procedure of Integer ME (IME) and Fractional ME (FME: include 1/2 pixel and 1/4 pixel precision ME). IME locates the best position of 16x16 pixel array in the global searching area of reference frame/filed that achieves best match of current MB

and reference frame/filed. FME further explore the local searching area around best IME position to find potential better match in the fractional-pixel interpolated frame/filed. After intra or inter prediction, integer transform & quantization are applied to reduce redundancy of prediction residual by reducing high-frequency information of residual values. Quantized residual coefficients, intra/inter prediction data (including prediction modes, reference frame/filed list, motion vector difference MVD), and coding control signals such as MB type, QP delta, and transform size flag are further compressed by the lossless entropy (statistical) coding to reduce redundancy of code words. An in-loop deblocking filter is allocated in the MB encoding feedback loop to reduce artifacts at the block edges of reconstructed frame/field. As the distortion of reconstructed reference frame/filed is reduced, deblocking filter can improve both subjective and objective visual qualities. Deblocking filter was applied as post processing stage in earlier standards, while it is integrated as an in-loop filter in H.264/AVC.

The MB decoding procedure of H.264/AVC is illustrated in Figure 1-1(b), including entropy (statistical) decoding, inverse quantization & inverse transform, MC or compensation of intra prediction, and deblocking filter. Computation complexity of decoding is significantly lower compared to encoding, because high complexity intra/inter prediction is not involved in decoding, and also because decoding mode of each MB is fixed according to MB type value; while in MB encoding procedure, multiple possible MB encoding modes need to be tested to select best MB coding mode and achieve better compression efficiency. The architecture of interpolation, reference frame/filed reconstruction and deblocking filter are same in both encoder and decoder. Computation complexity ratio of CABAC decoder in the video decoder is higher than

that of CABAC encoder in video encoder because of lower computation of other function blocks.

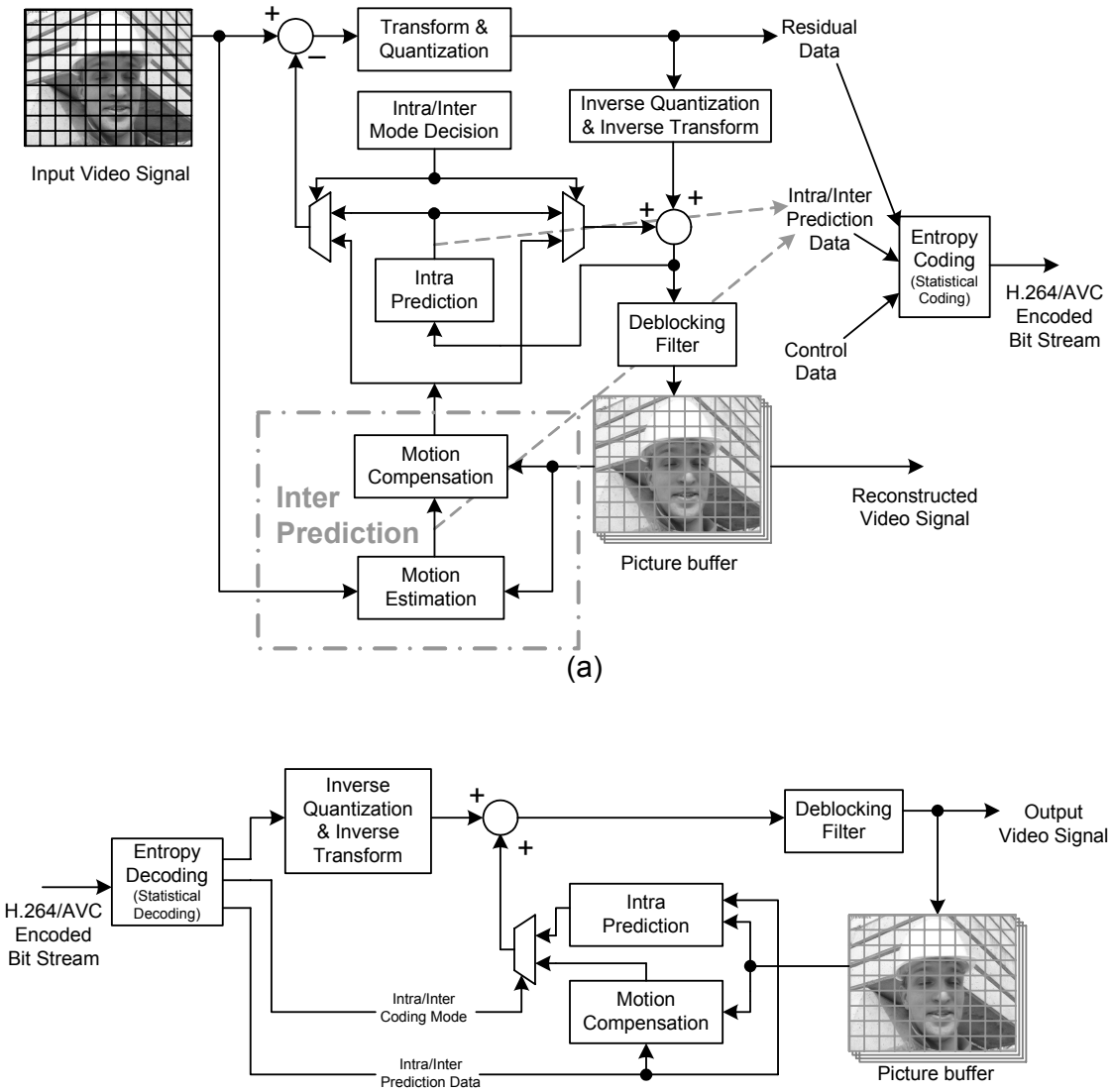


Figure 1-1: Block diagram of MB processing in H.264/AVC. (a) MB encoding, (b) MB decoding.

The significant improvement of compression efficiency of H.264/AVC [6] is attributed to several techniques, including adaptive Intra16×16/Intra4×4 intra prediction, multi-reference ME & MC, and variable block-size & 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 deblocking filter that enhances both subjective & objective video quality, more efficient entropy coding tools including CAVLC [9] and CABAC [10] compared to all previous standards, and Rate-Distortion Optimization (RDO) [11], etc. Moreover, adaptive frame/field coding at picture level (PAFF) and MB level (MBAFF) [8, 12] is beneficial in some scenarios, compared to frame coding or field coding.

Intra prediction: In the previous standards, intra prediction is always carried out in the transform domain, such as prediction of DC coefficients based on the neighboring coded DC coefficients in intra frame/fileds. In comparison, intra prediction of H.264/AVC is implemented in spatial domain, by referring to the neighboring pixels of previous coded blocks on the left and/or top of current predicting block. Four Intra-16×16 prediction modes are supported for block size of 16×16 and 9 Intra-4×4 modes are supported for block size of 4×4. Best prediction block size and prediction mode are chosen for each MB, and spatial redundancy is more efficiently reduced by coding the prediction error and prediction modes.

Integer transform: To remove redundancy in the transform domain, integer transform of H.264/AVC is used, which is an approximation of the DCT transform. The technique achieves exact match after decoding and the computation is also simplified, compared to the floating-point DCT transform in the other standards. More specifically, block sizes of 4×4 or 8×8 of integer transform can be adaptively chosen in the high level profiles of H.264/AVC to fit for various video scenarios. Small 4×4 transform is more locally adaptive and is required of transform region within small prediction Region [8]. After

redundancy reduction in the spatial and temporal domains, entropy coding tools are utilized to further remove the redundancy of code word.

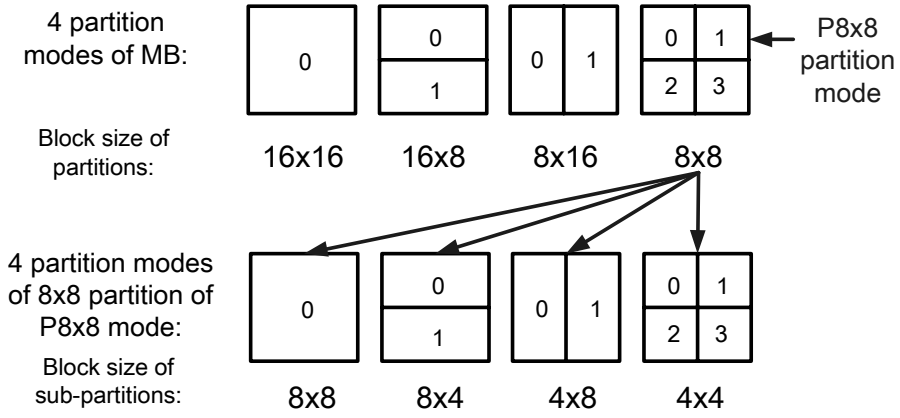


Figure 1-2: MB partition modes and sub-MB partition modes of ME in H.264/AVC

Inter prediction: The precision of inter prediction is enhanced compared to the earlier standards because of following technical improvements:

- Multi-reference inter-picture prediction allows encoder to select from a larger number of decoded and stored frame/fileds for motion compensation, compared to those of H.263 and MPEG-2. As a result, bit rate reduction is significant in certain types of video scene such as repetitive motion and back-and-forth scene.
- Variable block-size motion estimation of H.264/AVC supports more flexible selection of block size of motion compensation. As shown in Figure 1-2, except the 4 types of MB partition modes P16×16, P16×8, P8×16, and P8×8 of motion estimation with the corresponding partition sizes of 16×16, 8×16, 16×8, and 8×8 pixels that are supported in MPEG-4 Part 2, for the 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. The index numbers in the figure indicate scan and processing order of the partitions. It enables better match of various motion patterns and

more precise segmentations of motion regions, and results in bit-rate reduction of prediction residual data.

- The precision of motion estimation is ¼ of a pixel (quarter-pixel-precision or *qpel*), which is higher than that of most of previous standards. Interpolation operations using 6-tap FIR filter and bilinear interpolation are used to generate the pixels at half-pixel and ¼ pixel positions. The computation complexity of interpolation is lower than that of MPEG-4 Part 2.

Rate-Distortion Optimization (RDO): At MB level, coding efficiency depends on the selecting among different coding options. The best choice of coding options of MB achieves minimum distortion *D* within a constrained bit rate *R*. Instead of solving constrained selection problem, the widely used Lagrange multiplier methodology is applied, and the problem is transferred to a simpler unconstrained problem by finding the minimum (1-1), in which constant λ is the multiplier.

$$\begin{aligned} RD_{\cos t} &= D + \lambda \cdot R \\ \lambda_{\mathrm{mod}e} &= 0.85 \cdot 2^{\frac{QP-12}{3}} \end{aligned} \tag{1-1}$$

In H.264/AVC RDO algorithm, Lagrange optimization procedure of motion estimation of inter modes is separated from the successive procedure of MB coding mode decision. The multiplier λ_{mode} is used for MB mode decision, and it is a positive value proportional to the Quantization Parameter QP, as shown in (1-1). The multiplier λ_{motion} of motion estimation is set as the square root of λ_{mode}. For MB mode decision, coding modes are selected from intra and inter modes, including Intra-16×16, Intra-4×4, Skip, P16×16, P16×8, P8×16, P8×8, etc. Coding rate *R* and RD_{cost} of each MB coding mode are precisely evaluated, as all SEs of the MB are encoded by entropy coder CABAC or

CAVLC to obtain the accumulative value of R for MB coding mode. In comparison, the calculation of R is simplified in the procedure of best motion vector selection during motion estimation. The idea of RDO simplification of motion estimation was first proposed by Sullivan, *et al.* in [13] and updated in [11]. Because large amount computation involved in the evaluation of RD_{cost} values, R is approximated by a value proportional to the length of motion vector instead of going through entropy coding. A special case of RDO MB coding mode decision is mode P8×8, in which entropy coding is required to accurately evaluate the R of RD_{cost} for each sub-MB partition mode for the selection of best mode of each 8×8 sub-MB.

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) [9] and CABAC (context-based adaptive binary arithmetic coding) [10, 14]. In the Baseline and Extended profiles targeting at low bit-rate conversational network video service and stream services, CAVLC is utilized to encode 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 at 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 Chapter 2.

Although large percentage of H.264/AVC encoding computation is used for ME, throughputs (number of symbols coded per cycle) of both H.264/AVC video encoder and decoder are also limited by the entropy coding stage, because of sequential coding nature and high data dependency of CABAC coding procedure. As it is not efficient to remove

the bottleneck by software optimization and acceleration alone, it is reasonable to exploit parallelism at all levels to accelerate CABAC coding procedure in the H.264/AVC codec system targeting at high bit rate real-time coding.

1.2 Approaches of H.264/AVC Codec Acceleration

Because computation complexity of H.264/AVC is significantly higher compared to the previous standards, there has been much research on accelerating H.264/AVC encoding or decoding procedure in the aspects of embedded software implementation, algorithm modification and simplification, and hardware acceleration of codec system or particular function blocks by either FPGA or ASIC designs. For SW acceleration, DSP-based H.264/AVC encoder designs are reported in [15-18], while Cell processor [19] and ARM processor [20] are reported to achieve low-resolution SW decoding. Fast algorithms are developed to accelerate particular function blocks such as intra prediction [21], coding mode decision and RDO [12, 22-24], ME and MC [25], and rate control [26, 27]. However, SW acceleration is limited by the low degree of parallellism and is not suitable for high bit rate high definition real time coding.

Hardware acceleration of H.264/AVC codec is reported in the literatures targeting at encoder/decoder system or particular function blocks. For encoder design, MB encoding is accelerated by 4-stage pipeline [28, 29] or 3-stage pipeline [30] to enable parallel processing of different MB coding steps such as integer ME, fractional ME, transform & quantization. To remove data dependency and enable pipelined coding, algorithm is adjusted, including simplified MV prediction in [28, 29]. Different encoding stages are controlled by embedded processor [31] or through control signals input from system bus interface [30]. As computation complexity of decoder is significantly lower, FPGA is

utilized to achieve real time decoding excluding entropy decoding in [32]. Schemes of memory access reduction and memory size reduction of decoder are reported with strategies of optimized scheduling of decoding order [33], data reuse by allocation of shared memory and local buffers [28-30, 34], and multi-bank SRAM access [35]. Power reduction and chip testing schemes of codec are considered in [36, 37].

HW designs that focus on accelerating of particular function block are also reported. To reduce ME computation, MB partition modes and search candidates are reduced in [30], full search early termination of ME is applied in [34], and control of search range and reference frame number in [38] according to input variations. However, video quality is also degraded [39] with such simplification. SIMD architecture of ME is designed in [40] to enhance computation parallelism. For MC of decoder, interpolation window reuse scheme [41] 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 [42] and scheduling of parallel processing of Intra16×16 & Intra4×4 prediction [43].

HW acceleration of entropy coding stages at H.264/AVC is necessary because the bottleneck of strong data dependency and sequential coding property can not be efficiently removed by SW design and optimization. HW architectures of CABAC and CAVLC codec designs and related design strategies will be analyzed in Chapter 3.

1.3 Objectives of the Research

As aforementioned, the entropy coding tool CABAC exhibits outstanding efficiency of lossless compression compared to CAVLC and other VLC encoders and contributes significantly to the performance enhancement of H.264/AVC. However, sequential

coding nature and strong data dependency of CABAC coding procedure prevent efficient software acceleration in both single-core and multi-core parallel coding at MB level. Although multi-core parallel coding can be applied at slice level, compression efficiency of CABAC will be degraded when a frame/filed is divided into multiple slices. Quite a number of research projects have been carried out targeting at hardware design of CABAC encoder of H.264/AVC standard in recent years. Although different approaches have been investigated to accelerate the encoding procedure, these designs still have limitations in several aspects, including incomplete functional implementations, inefficient removing of the dependency of coding data, no support of RDO coding in the CABAC encoder, and high frequency of memory access for the context model and related high power consumption.

Because CABAC is the final encoding stage of video encoder and the first decoding stage of video decoder of H.264/AVC, it has significant influence on the coding performance of the top-level video codec. Furthermore, because the processing data rate at CABAC encoder is significantly higher compared to that of decoder, especially when RDO is used in the coding control procedure, it is challenging to design a real-time CABAC encoder targeting at high definition high quality H.264/AVC video coding applications.

In this thesis, research work is carried out to design a hardware IP of CABAC encoder targeting at the Main profile of H.264/AVC. The general research objectives include:

(1) Design a SoC based full hardware CABAC encoder that minimizes computation on the host processor and data transfer on system bus. (2) Enhance throughput of encoder and achieve high quality real time video coding. (3) Provide a solution of SoC-based CABAC encoder IP with complete RDO support, and insure integratability and

reusability, and wide application field. (4) Minimize memory access frequency and power consumption of encoder. (5) Explore general circuit design methodologies (strategies) that can be used for sequential coding algorithm and system such as entropy coding.

1.4 List of Publications

- X.H. Tian, T.M. Le, X. Jiang, and Y. Lian, "Full RDO-Support Power-Aware CABAC Encoder with Efficient Context Access," *IEEE Transactions on Circuits and System for Video Technology (T-CSVT)*, vol. 19, no. 9, pp. 1262-1273, Sept. 2009.
- X.H. Tian, T.M. Le, X. Jiang, and Y. Lian, "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, 2008.
- X.H. Tian, T.M. Le, X. Jiang, and Y. Lian, "Implementation Strategies for Statistical Codec Designs in H.264/AVC Standard," in Proceedings of *The 19th IEEE/IFIP International Symposium on Rapid System Prototyping*, pp.151-157, 2008.
- X.H. Tian, T.M. Le, H.C. Teo, B.L. Ho, and Y. Lian, "CABAC HW Encoder with RDO Context Management and MBIST Capability," in Proceedings of *International Symposium on Integrated Circuits*, pp.236-239, 2007.
- X.H. Tian, T.M. Le, B.L. Ho, and Y. Lian, "A CABAC Encoder Design of H.264/AVC with RDO Support," in Proceedings of *18th IEEE/IFIP International Workshop on Rapid System Prototyping*, pp.167-173, 2007.

- T.M. Le, X.H. Tian, B.L. Ho, J. Nankoo, and Y. Lian, "System-on-Chip Design Methodology for a Statistical Coder," in Proceedings of *Seventeenth IEEE International Workshop on Rapid System Prototyping*, pp.82-90, 2006.
- Patent: US Provisional Application No. 61/151,269. Title: Method and Device for Encoding Syntax Element using CABAC Encoder. Filing Date: 10 February 2009
- X.H. Tian, T.M. Le, and Y. Lian, *Entropy Coders of the H.264/AVC Standard – Algorithms and VLSI Architectures,* Springer-Verlag GmbH, Publisher in editing procedure, Nov. 2009.
- X.H. Tian, T.M. Le, and Y. Lian, "Analyses on the Implementation Techniques of CAVLC and CABAC Codecs in H.264/AVC," *IEEE Transactions on Multimedia,* 2009. (Journal submission under review)

This research is restricted to the efficient design of CABAC encoder, and the other functional blocks of H.264/AVC standard are not implemented in hardware circuits. The thesis is organized as follows. Arithmetic coding theory and CABAC algorithm are introduced first in Chapter 2. After that, related literatures on H.264/AVC entropy codec designs are reviewed in Chapter 3. The proposed CABAC encoder design of this thesis is introduced in Chapter 4 and Chapter 5. Functional partitioning schemes, top-level HW encoder architecture, and part of function blocks of encoder are discussed in Chapter 4, while the architecture of context modeling is discussed in Chapter 5. Then the design of the SoC system bus interfaces and inter-connection of the encoder is described in Chapter 6. After that, design, synthesis, verification, and performance comparison to the reported designs are illustrated in Chapter 7. Conclusions are given in the last chapter.

Chapter 2 Review of Arithmetic Coding and CABAC

2.1 Introduction of Arithmetic Coding

Compared to the previous lossless variable length coding (VLC) methods [44] including Elias, Golomb and Rice, Shannon-Fano, and Huffman [45], the distinct difference of arithmetic coding is that code words can be represented using fractional number of bits, while in other VLCs, each code word must occupy integer number of bits. Shannon first mentioned the possibility of such coding method in 1948 [46]. Elias explores the idea of successive subdivision of coding interval [47] in 1960s. Complete scheme of arithmetic coding was proposed by Rissanen [48] and Pasco [49] independently in 1976, in which finite-precision arithmetic coding was implemented. Further research work include hardware-oriented arithmetic coders [50] of IBM and software-oriented arithmetic coders by Witten *et al.* [51], which made it practical in the image and video compression applications. Arithmetic coder generates code word by representation of subintervals of the interval [0, 1) with enough bits. Ratio of each subinterval to the current interval is proportional to the probability of the corresponding event. If only two events (symbols) are coded, 1 bit is enough to represent *most probable symbol* (MPS) and *least probable symbol* (LPS), and it is called binary arithmetic coding and each coding symbol is called a bin. Context-based adaptive binary arithmetic coding is binary arithmetic coding with adaptive symbol probability according to the recent coding events.

As shown in Figure 2-1, coding interval of binary arithmetic coding can be defined as [Low, Low + Range). For each bin encoding, the interval is subdivided into two

subintervals [Low_{LPS}, Low_{LPS} + $Range_{LPS}$) and [Low_{MPS}, Low_{MPS} + $Range_{MPS}$), and one subinterval is selected based on whether the coding bin is MPS or LPS. Confirmed bits of Low are output as coding result. Subinterval calculation of MPS and LPS is according to (2-1), in which Range of LPS ($Range_{LPS}$) is calculated according to p_{LPS}, the probability that the coding bin is LPS. Low is updated accordingly after Range update.

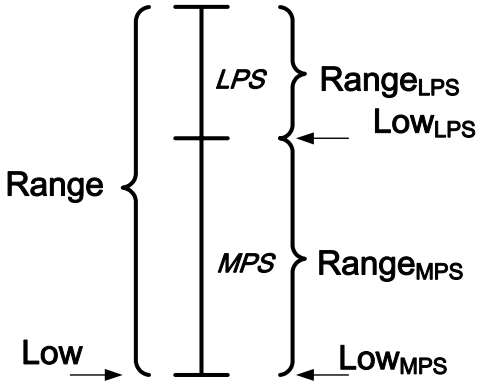


Figure 2-1: Coding interval subdivision of binary arithmetic coding.

$$\begin{aligned} Range_{LPS} &= Range \times p_{LPS} \\ Range_{MPS} &= Range - Range_{LPS} \\ Low_{LPS} &= Low + Range_{MPS} \\ Low_{MPS} &= Low \end{aligned} \tag{2-1}$$

As coding interval is represented by finite number of bit, to overcome precision loss introduced by the shrinking of current interval and achieve incremental encoding and decoding procedure, an incremental output method of arithmetic encoding is proposed in [51], in which the interval is upscaled by left-shift of Range and Low when Range of interval is less then ¼ 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.

2.2 CABAC of H.264/AVC

CABAC stands for Context-based Adaptive Binary Arithmetic Coding. Although Q-coder [52], QM coder [53], and MQ coder [54]of previously image coding standards are also binary arithmetic coders with statistical adaptivity, CABAC is first proposed by Marpe *et al.* in 2001 [55] as a proposal to the H.264/AVC standard committee. It is adopted as the entropy coding tool used in the Main profile and High profiles of H.264/AVC standard. Before CABAC of H.264/AVC, LUT(lookup table)-based variable-length coding (VLC) are generally utilized for entropy coding in the hybrid block-based video coding standards including H.263, MPEG-2, MPEG-4 Part 2, *etc*. The limitation of VLCs [10] is that coding event with probability higher 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. The only arithmetic coder adopted in video standard is of Annex E of H.263 [4], in which coding efficiency of entropy coding is not significantly improved, because of directly using of SEs of VLC for arithmetic coding without redefinition. Before CABAC proposals of [55-57] of H.264/AVC, similar arithmetic coding approaches were first investigated and applied in non-block-based video coding [58, 59], such as DWT.

CABAC [6, 10, 14] of H.264/AVC is the first successful arithmetic coding scheme deployed in video coding standard, with significant compression improvement compared to previous entropy coding tools. As shown in Figure 2-2, CABAC encoding process

consists of three elementary *steps*: binarization, context modeling, and binary arithmetic coding (BAC). 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, context model (probability model) of the bin is prepared by the step of context modeling. Techniques of the three steps will be discussed in the following subsections.

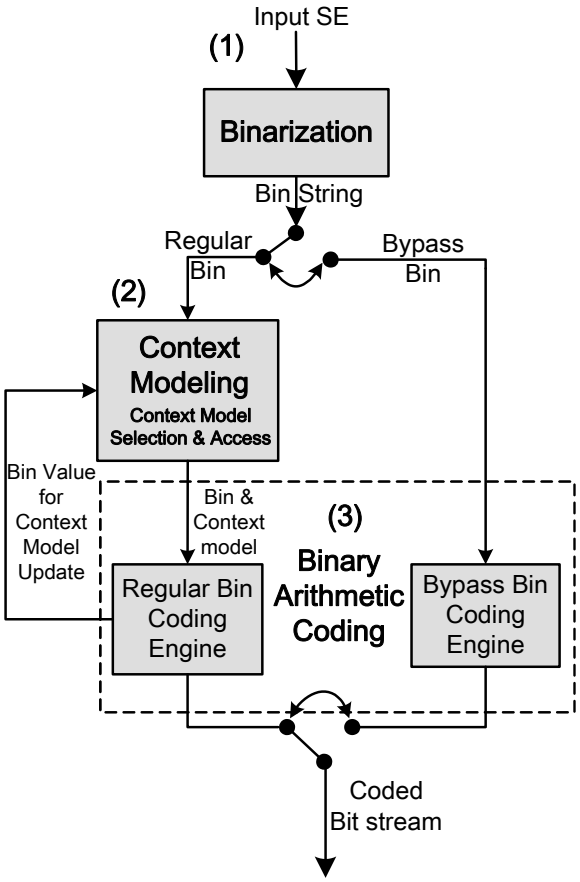


Figure 2-2: Block diagram of CABAC encoder [6] of H.264/AVC.

2.2.1 Binarization

Binarization maps non-binary valued syntax elements (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 [10] 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 binary arithmetic coding can be utilized; (c) context modeling at sub-symbol (sub-SE) level provides more accurate probability estimation than context modeling at symbol level, and the alphabet of 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). K^{th} order Exp-Golomb binarization (EGk) [60], a derivative of Golomb coding [61], is proved to be optimal prefix-free coding for geometrically distributed sources. EGk code word consists of prefix and suffix bin strings, with total length of $2l+k+1$ bits. EGk prefix is a Unary code word, with l bits of 1 and one terminating bit 0). The length l of string of bit 1 is represented as:

$$l = \left\lfloor \log_2\left(\frac{x}{2^k}+1\right)\right\rfloor \tag{2-2}$$

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

$$EGk\_suffix = x + 2^k - 2^{k+l} \tag{2-3}$$

UEGk is combinational 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 utilized to reduce computation complexity.

2.2.2 Context Modeling

The context modeling step shown in Figure 2-2 implements two aspects of functions: context model selection & context model access. The statistics of coded SEs are utilized to update probability models (context model) [8] of regular bins. For regular bin coding, one context model is chosen and fetched from a pre-defined set of context models to provide probability of regular bin to be MPS or LPS, and context model is updated after bin coding based on bin value. 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 sufficient limited set of representative values, and 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 1-bit tag of MPS value and a 6-bit *pStateIdx* (probability state index) 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 (2-4). The ratio of two neighboring probability values is a constant value α, which is approximated to 0.949.

$$p_\sigma = \alpha \cdot p_{\sigma-1}$$
$$for: \sigma = 1,\ldots,63, \qquad \alpha = \left(\frac{0.01875}{0.5}\right)^{1/63}, and \qquad p_0 = 0.5 \tag{2-4}$$

Probability update of context model is based on the rule in (2-5), 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.

$$p_{new} = \begin{cases} \max(\alpha \cdot p_{old}, p_{62}), & if \quad bin = MPS \\ 1 - \alpha \cdot (1 - p_{old}), & if \quad bin = LPS \end{cases} \tag{2-5}$$

By mapping the update probability value of LPS of (2-5) 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 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 (2-1) 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 methods are supported in CABAC, based on (a) neighboring coded SE values of 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.

2.2.3 Binary Arithmetic Coding (BAC)

The step of binary arithmetic coding performs arithmetic coding of each bin based on bin value, type, and corresponding context model of the bin. BAC is a recursive procedure of coding interval subdivision and selection, as shown in Figure 2-3.

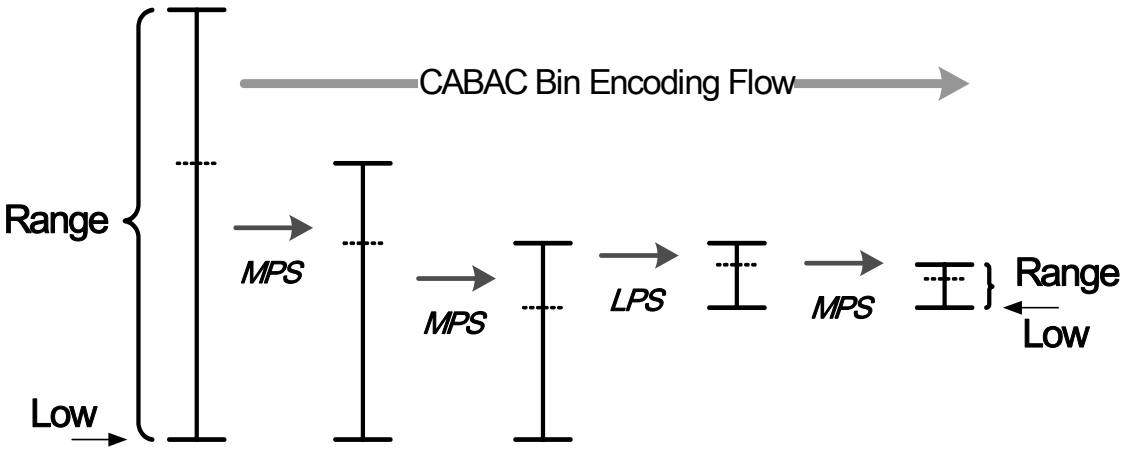


Figure 2-3: Coding interval subdivision and selection procedure of CABAC.

Coding interval subdivision mechanism of CABAC is different from that of QM and MQ coders. In QM and MQ coders, calculation of $Range_{LPS}$ of (2-1) is simplified by using approximated value 1 of Range, and the multiplication is removed. In comparison, Range is also utilized for $Range_{LPS}$ calculation of CABAC. Figure 2-4 shows the reference pseudo C program of interval subdivision and selection of regular bin, in which 2 higher

bits of Range (bit 7 and bit 6) and the index of probability state (pStateIdx) of LPS are used to lookup pre-calculated product of Range and p_{LPS} of (2-1) from a 2-dimentional LUT. Although the product of LUT is of with limited precision, precision of $Range_{LPS}$ calculation and interval subdivision is improved and computation complexity is minimized in CABAC, compared to that of QM and MQ coders.

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

Figure 2-4: Coding interval subdivision and selection of regular bin of CABAC.

```
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;
}
```

Figure 2-5: Pseudo-C program of renormalization and bit output of CABAC.

Because Range and Low of coding interval are represented by finite number of bits (Range: 9 bits, Low: 10 bits), 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 algorithm of [51], as illustrated in the reference pseudo C program of Figure 2-5. 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 Figure 2-5, renormalization of Range and Low is an iterative procedure, and the maximum iteration number is 6, as the smallest possible value of Range is 6. For the processing of carry propagation and output of coding bits, coded bits of CABAC are not output until it is confirmed that further carry propagation will not influence bit values. Figure 2-6 illustrates that only when interval length (Range) is smaller than 0x100 (threshold), one bit can be output if the interval is located within the top half [0x200, 0x400) or bottom half [0, 0x200) of maximum coding range, or an 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 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 bit stream, compression efficiency of CABAC is further improved. However, renormalization illustrated in Figure 2-5 is a highly sequential operation, and as the iteration number is variable depending on selected subinterval Range, it is challenging for SW or HW

acceleration of renormalization and bit output of BAC. In some situations, long delay can be caused when large number of OS bits are accumulated.

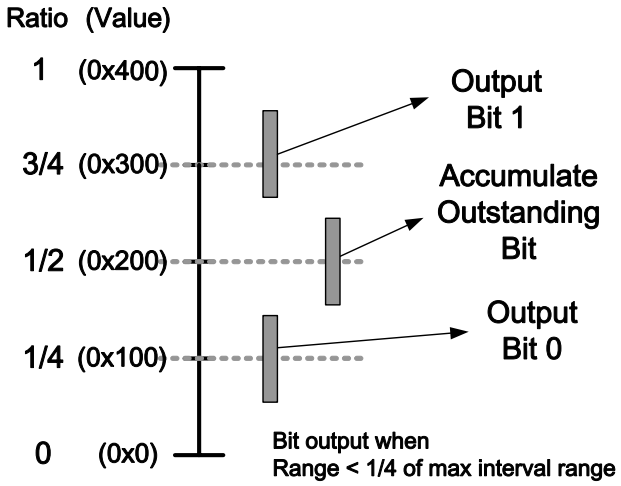


Figure 2-6: Decision of bit output and accumulation of outstanding (OS) bit.

2.2.4 Comparisons of CABAC with Other Entropy Coders

Coding efficiency of CABAC is higher compared to that of the other arithmetic coders including Q-coder, QM coder, and MQ coder that are in the earlier image processing standards, because (a) more precise approximation of multiplication of $Range_{LPS}$, (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 high computation complexity of CABAC, another entropy coding tool CAVLC [9] is deployed in the Baseline profile and Extended profile of H.264/AVC targeting at low bit-rate real-time video coding. It offers compression-complexity tradeoff with lower coding efficiency and lower complexity compared to CABAC [10]. It is employed to encode quantized transform coefficients of 4×4 residual blocks, while zero-

order Exp-Golomb codes [60] (EG0) are used for all other types of non-residual SEs. Adaptivity is introduced to CAVLC by switching among multiple VLC tables based on already processed SEs, and coding efficiency of CAVLC is better than the previous VLC coders with single VLC table. Instead of coding data pair of run-level as single SE, run and level of 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 [10], compared to CAVLC & 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.

Chapter 3 Review of Existing CABAC Designs

Since the adoption of CABAC entropy coding scheme in H.264/AVC [10, 14, 58, 59, 62, 63], CABAC is also applied in many applications of image and video processing including motion mode and residual data of 3D dynamic mesh [64], prediction residual in lossless 4D medical image compression [65], SEs of 8×8 transform coefficients of AVS coding standard [66], motion vector coding of scalable video coder [67], parameters of depth and correction vectors in multi-view video coding [68]. CABAC is also utilized to encode affine motion vector [69], and MVD of 3-D DWT-based subband video encoder [70].

Algorithm optimization of CABAC of H.264/AVC is also carried out targeting at enhancing accuracy of the context model selection in MVD coding [71], investigating parallel CABAC coding using table lookup technique with parallelized probability models [72], analyzing error detection probability (EDP) of CABAC coded SEs [73], error resilience enhancement of coded bit stream by inserting detective markers based on CABAC semantics [74], or error detection based on joint source-channel MAP estimation [75, 76].

Recognizing the highly computational complexity of motion estimation, because of the sequential coding nature and high data dependency of coding procedure in 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 software optimization and acceleration alone, a number of hardware designs for CABAC have been proposed, to enhance throughput in

various applications. In the following sections, different implementation strategies of CABAC encoding and decoding architectures will be investigated. The strategies are evaluated using circuit area, processing time, and power consumption as judging criteria. The strategies are also investigated at the video codec level in terms of host computational complexity, data transfer on system bus, and total memory/buffer usage. The suitability of strategies is evaluated in different application scenarios such as low power or high speed application. Discussion and analysis of technical advantages and limitations of these implementations are beneficial for the further design of high performance entropy codec in various image and video processing applications.

3.1 CABAC Decoder and Encoder IP designs of H.264/AVC

CABAC achieves higher compression efficiency compared to CAVLC. CABAC encoder and decoder IPs in the recently reported literatures are reviewed as follows. Benefits and limitations of the implementation strategies of these designs are discussed and analyzed.

3.1.1 CABAC Decoder Designs

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

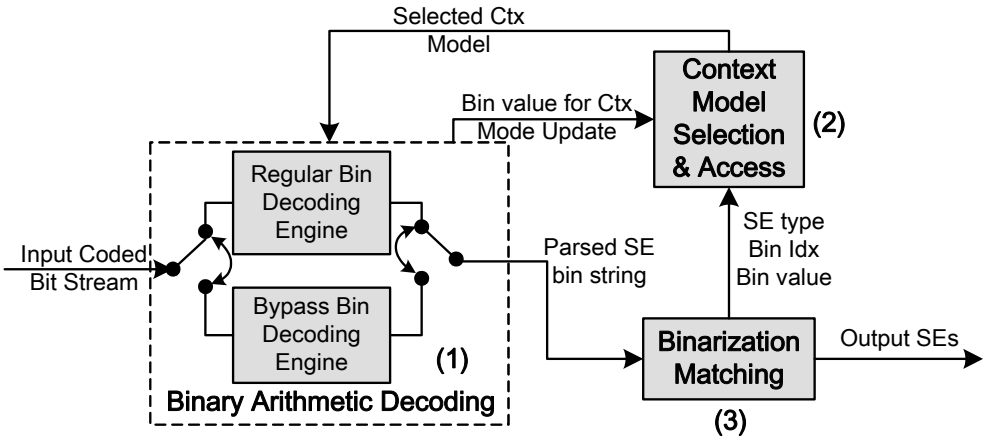


Figure 3-1: Block diagram of CABAC decoder.

As shown in Figure 3-1, 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 CM based on the decoding SE type, bin index (*binIdx*) and decoded bin values provided by BM and SE values of decoded SEs of neighboring decoded MBs. Each context model can be accessed according to *CtxIdx*, which is the sum of context offset (*CtxOffset*) and 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. A long loop

of data dependency exists to select a proper context model for next RB (BAD -> BM -> CM -> BAD). Design challenge of CABAC decoder is to break the loop and enhance decoding throughput.

(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 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 can be achieved with throughput of 2 bin/cycle (2 RB, 2 BB, or 1 RB and 1 BB) [77-79]. 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 [77], and the correct R_{LPS_4} value is selected for RB2 when RB1 is decoded. Dual-RB decoding of [80] constantly predicts that RB1 is MPS, and begins RB2 decoding, and the critical path of RB decoding is shorter than that of [77]. However, throughput of [80] is only 0.56 bin/cycle, because if the prediction is wrong, 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 throughput of 1 bin/cycle including non-residual SEs and coded block flag (CBF), decoding time is prolonged because of longer critical path length. 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.

(2) Context Model Selection & Access

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 context RAM access delay, a group of context models can be prefetched to the local buffers [77, 81]. Because the context model selection and access of coded block pattern (CBF) requires multiple cycles [82], *CtxIdx* of the next CBF is pre-calculated during decoding of current 4×4 block [79]. 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 it is possible to simultaneously access context models of both SCF and LSCF SEs [79]. The techniques of context model prefetch, *CtxIdx* pre-calculation, and parallel access of multiple context models are beneficial to reduce context access delay.

(3) Binarization Matching and SE Generation

BM is the inverse binarization of CABAC encoding. It can be controlled by a FSM to identify the type of next decoding SE. The corresponding LUT of the SE is accessed to match the decoded code word. The parsed SE is output in this step. Because BM is one decoding stage of CABAC decoding pipeline, several strategies are discussed in subsection 4) to reduce pipeline stall and enhance decoding throughput, including parallel processing of BAD and BM.

(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 [83]. The acceleration strategy is to map sequential operations of memory access and table lookup of CA & BAD into pipeline stages with data prefetching and data forwarding. 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 removed. Actual throughput of decoder is influenced by both the maximum processing speed of CA & 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 [81], which is also adopted in [84] and [85]. 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 regular bin decoding. A small context buffer named CMR is also allocated in [81] that can buffer 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 [81] 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 [81] 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 3.93 cycles to decode one bin. In the updated design [85] of [81], 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 [85], over 60% decoding cycles are reduced compared to [81]. 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, an architecture of 4-stage pipeline is reported in [84] that achieves decoding throughput of 1 bin/cycle for RB decoding. Context model selection, and CL are separated to stage 1 and stage 2 of the pipeline. During decoding of current SE, the first context model of next SE is selected in stage 1 and loaded from context RAM in stage 2. In stage 2, two context models are read separately from two context RAMs in parallel per cycle. One context model is of current SE and is read from one RAM containing entire context models, while the other context model is of the next SE and is read from a small RAM that is not allocated in the other designs. The decoded bin in stage 3 of BAD & 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. By prefetching context model of next SE during current SE decoding, pipeline stall in the case of SE change is avoided in [84] compared to the other designs. Decoding speedup of 6.6 times is achieved 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. Power consumption and circuit area are relatively increased.

3.1.2 CABAC Encoder Designs

Block diagram of CABAC encoder of H.264/AVC is shown in Figure 2-2. CABAC encoder consists of 3 functional steps: (1) *Binarization* to map SE value to bin string; (2)

Context model selection & 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. Additionally, 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 stall compared to previously discussed decoder designs. CABAC encoder designs of H.264/AVC are reviewed as follows.

Reported by Li *et al.* [86], 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 it only achieves a throughput of 0.59 bin/cycle.

A CABAC encoding core of context access and BAC is proposed in [87] 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. Low power technique such as clock gating is also adopted to reduce power consumption of the tag cache. 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 [88]. To encode 2 RBs per cycle, the inverse multiple branch selection (IMBS) is adopted and parallelism of *Range* update is achieved by pre-calculating all 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 given in [88].

In [86, 87], binarization and context model selection are left to be run on the host processor. This implies additional processing power and memory bandwidth must be supported by the higher level system.

Chen *et al*. [89] combines the *CtxIdx* (context index) calculation and binarization into one module of a 3-stage pipeline. Similar to [88], the 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.* [90] 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, encoding pipeline still stalls 4 cycles for each access. The resulting throughput of the encoder is 0.67 bin/cycle.

A full hardware CABAC encoder is also proposed by Lo *et al.* [91] with 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. Because critical path length of BAC is prolonged, 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. Additionally, because only one context model of *level* can be accessed per cycle, 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 [92]. 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. Additionally, cascaded units coding

interval subdivision and selection are needed and the critical path length is significantly longer compared to single bin coding scheme. 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 [93] that the high computational requirement of CABAC is largely due to the support of RDO in the H.264/AVC encoding system. Design of CABAC encoder with RDO functionality is considered in [93], with supports of context state backup & restoration operations when RDO mode changes. 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 throughput of FIFO read or write is single context model per cycle, timing delay is long for the operations of context state backup and restoration. Additionally, large FIFO buffers occupy large circuit area. Because single 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 is significant. In [93], 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 [93] 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 [94]. The coding throughput is

further enhanced in their latest design [95], in which *Range* & *Low* are updated in separate arithmetic coding pipeline stages with duplicated computational resources to significantly increase throughput to around 2 bins/cycle. 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, context models are prefetched 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. However, these operations take up 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 [95], which is critical to the efficiency of inter frame coding of H.264/AVC. In Chapter 7, limitations of [95] will be analyzed in details.

3.2 Summary of Implementation Strategies of Entropy Codecs

Hardware design implementation strategies of entropy coding tools CABAC of H.264/AVC have been investigated. The strategies are evaluated using criteria of circuit area, processing time, power consumption, etc. 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.

For CABAC encoder designs, the implementations discussed above 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, a dedicated CABAC encoder IP of H.264/AVC standard with comprehensive coding functions and stable high performance is proposed to solve the limitations aforementioned and achieve research objectives listed in Chapter 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.

Chapter 4 The Proposed Design of Hardware CABAC Encoder

In this chapter, architecture of the proposed hardware (HW) CABAC encoder is illustrated. Firstly, the design methodology of a SoC (System-on-Chip) based entropy coder is presented. Secondly, based on the design methodology, hardware/software (HW/SW) functional partitioning of CABAC encoder function is carried out to decide proper portion of functions to be designed as HW circuits and the portion of functions remained 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 following sections, design details of major function blocks are presented including binarization & bin packet generation and binary arithmetic coding (BAC). Moreover, additional functions supported by the encoder are discussed including context model initialization, RDO function support in BAC, etc.

4.1 Design Methodology of SoC-based Entropy Coder

Design methodology for a SoC-based entropy coder such as CABAC is proposed [96], in which the entropy coder is realized as an IP block that can be integrated in a SoC video coding system. The IP block can be a logic synthesizable RTL design or a hard IP block where physical implementation is done and GDSII is ready. The design flow, as shown in Figure 4-1, contains 9 elementary 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 synthesis, introducing SoC features, and HW/SW co-simulation.

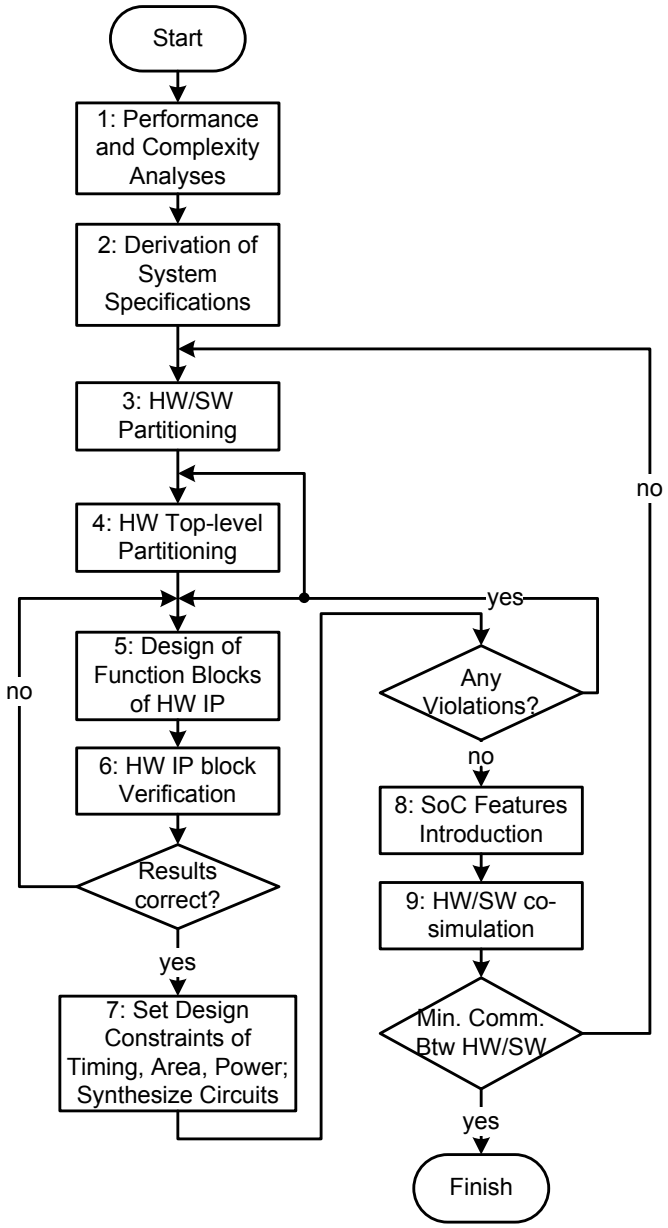


Figure 4-1: SoC-based entropy coder design flow.

The first step of design flow is performance and complexity analyses. It is crucial 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 microprocessor. If the complexity is high, it is better to

identify the bottlenecks. After complexity analysis is performed, system specifications are written in step 2 to address not only user's needs but also plans to minimize the effects of bottlenecks. In step 3, 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 embedded microprocessor (host processor). The decision by which HW/SW are partitioned is based on the system specification such as coding speed of entropy coder, and constraints of communication latency between host process and 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*. In step 4, the function of the *HW IP block* is further partitioned into proper function blocks during top-level HW architecture design, which is crucial to the performance of HW IP. All function blocks of HW IP are designed accordingly in step 5, followed by top-level HW IP block verification in step 6. 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. The RTL-level functional correct design is constrained with timing, area, and power constraints and synthesized into gate-level circuits in step 7. If any of the constraints are violated, function blocks are redesigned in step 5. If design violations still exist after critical function block redesign, the design flow will go back to step 4 to adjust the top-level entropy coder architecture with more proper 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. 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 FIFO 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. Co-simulation of both HW IP block and SW non-IP block is done in step 9 to verify if communication between SW non-IP block and HW IP block is minimized including data transfer latency and system bus bandwidth occupation.

4.1.1 Performance & 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 [97] in the QP (quantization parameter) range of 28 to 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.

Results of Table 4-1 illustrate 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 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.

Table 4-1: H.264/AVC encoder bit rate reduction, with CABAC compared to with CAVLC

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

Instruction-level program analyzing tool PIN [98] is used to utilized for profiling reference JM encoder, and it is found that the algorithm in CABAC reference SW is more computational intensive compared to that of CAVLC & Exp-Golomb code of Baseline profile, partly due to the additional instructions executed for binarization and context modeling (context model selection and access from memory). The higher computational demand required by CABAC is reflected in the video encoder when RDO is employed, as large number of intra and inter prediction modes are tested during MB coding mode decision. Compared to CAVLC & 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. High computation and bandwidth make it difficult for CABAC reference SW to meet requirements of real-time encoding. Therefore, it is 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 of JM reference H.264/AVC encoder. All CABAC related functions are classified into 5 categories, including C1 to C5. The details of CABAC function of each category is shown in Table 4-2.

Table 4-2: Five function categories of CABAC encoder of instruction-level analysis

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

PIN tool is utilized to monitor 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 shown in Figure 4-2 and Table 4-3.

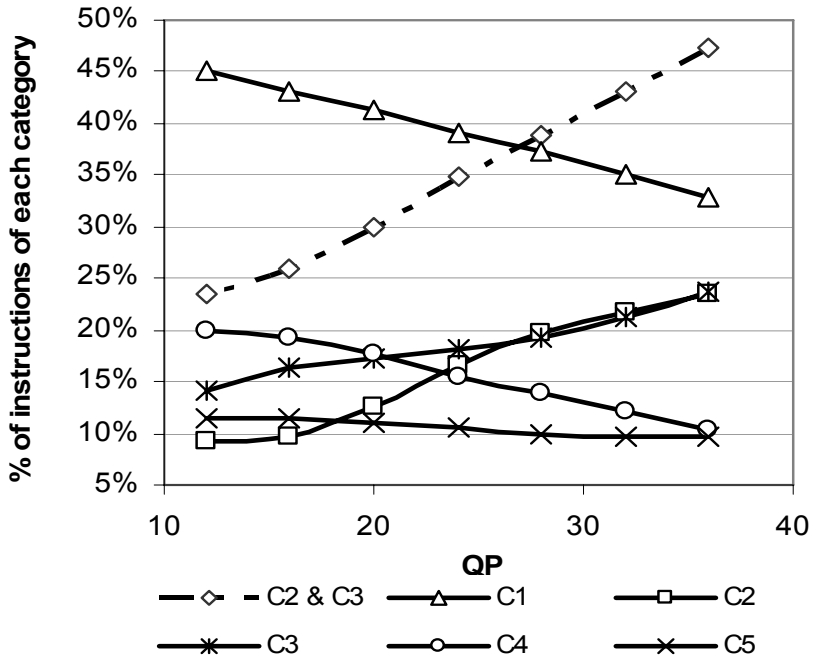


Figure 4-2: Five CABAC functional categories as % of total CABAC instructions in CIF test of H.264/AVC encoder of JM reference SW in the QP range of 12 to 36.

Table 4-3: Percentage of instructions of each category of CABAC encoding function in CIF sequence analysis

Category / QP	C1	C2	C3	C4	C5	C2 & 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%

Table 4-4: Percentage of instructions of each category of CABAC encoding function in HDTV 720p sequence analysis

Category / QP	C1	C2	C3	C4	C5	C2 & 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%

From Figure 4-2 and Table 4-3, it can be observed that CA & BAC (C1) only occupies 32.9% to 45.2% of total CABAC instructions, and the percentage number decreases when QP increases. CBF coding excluding computation of CA & BAC (C3) occupies 14.2% to 23.8% of total CABAC instructions, because of high data traffic and complex procedure of context model selection (CtxIdx calculation) of CBF. The remaining computation (C4) of SCF, LSCF, and level, excluding CA & BAC occupies 10.3% to 19.9% of total

CABAC instructions. The mapping procedure of run-level pairs to 4×4 block coefficients (C5) takes up 9.6% to 10.5% total CABAC instructions. The sum of % instructions of C2 and C3 is demonstrated as dashed line in Figure 4-2, representing the sum of non-residual SE and CBF coding (excluding CA & BAC). This sum occupies 23.5% to 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. 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 & C3 instructions increases from 34.8% (CIF) to 36.3% (HDTV 720p).

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. System specification (step 2 of SoC-based entropy coder design) is derived before HW/SW functional partitioning, and detail targets of the design are specified. CABAC encoder IP design is required to achieve real-time HDTV encoding at Main profile of H.264/AVC, support different video coding configurations including 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 specifications, different HW/SW partitioning schemes of CABAC

encoder are analyzed and evaluated. Furthermore, necessity of supporting RDO functions in CABAC encoder IP is emphasized.

4.2.1 Analysis of Different Partitioning Schemes

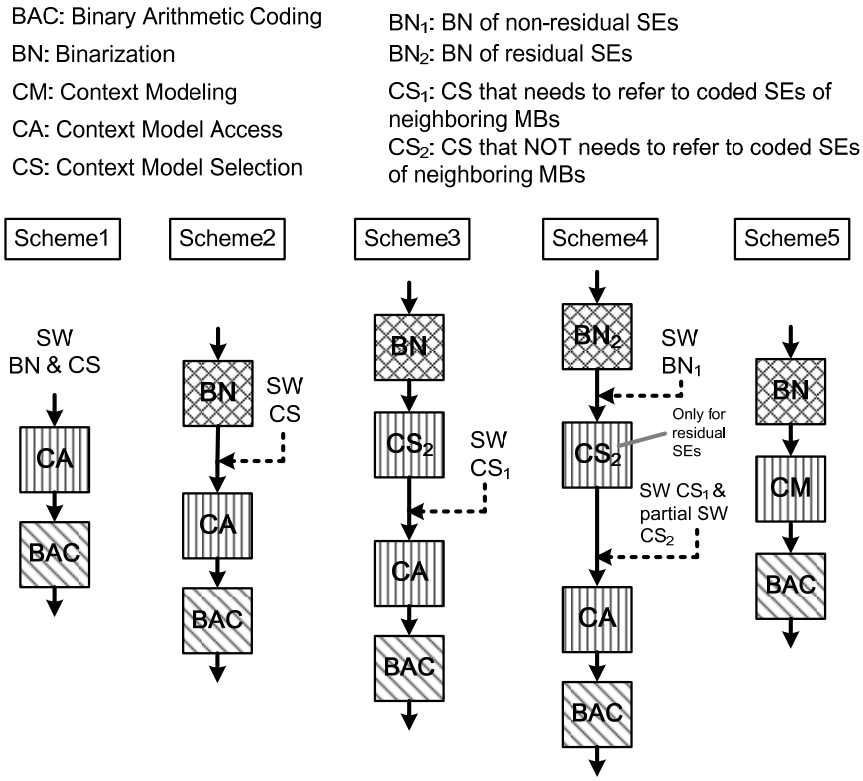


Figure 4-3: Five schemes of HW/SW partitioning of CABAC encoding.

In the 3 elementary 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 go through steps of BN, CS, CA, and BAC. CS can be further partitioned 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 proposed design of this thesis are summarized and analyzed as follows, and in Figure

4-3, the functions partitioned to HW implementation of each scheme are highlighted as blocks, while the functions partitioned to SW are shown as text containing "SW".

Scheme 1: HW Design of Context Model Access (CA) and BAC

As shown in the figure, the HW partition only contains CA and BAC, while the remaining function of BN and CS are left to be completed by SW. This is the scheme adopted by several designs including [86, 87]. In some designs [93, 99, 100], CA is also partitioned to 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. For this scheme, remaining computation on host processor is even higher than HW IP. As shown in 4.1.1, CA & BAC occupies only 39.2% of CABAC instructions. System bus bandwidth is also high because the bin value and context model index (CtxIdx) of each regular bin need to be transferred through system bus. In general, CABAC encoding acceleration is not significant in scheme 1.

Scheme 2: HW Design of Binarization, CA, and BAC

In this scheme [101], binarization is also partitioned to HW, compared to scheme 1. Only context model selection is partitioned 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. Therefore, binarization is also required in SW. The improvement of this scheme compared to scheme 1 is that CABAC packet size can be reduced as SE values are transferred instead of bin strings. However, as computation resources of HW and SW are overlapped in this scheme, CABAC encoding is not sped up compared to scheme 1.

Scheme 3: SW Implementation of Context Model Selection of CS_1

In this scheme, only when coded SEs of neighboring blocks/MBs are refer to select context model (calculation of CtxIdxInc), the CS function is partitioned to SW (shown as CS_1 in Figure 4-3). This scheme is adopted in our previous work [102], because compared to the scheme 1 and scheme 2, large ratio of CS operations are removed from 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 1st 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_1 is complex and inefficient in SW. Profiling results of 4.1.1 illustrate that computation of C2 & C3 occupies over 1/3 of CABAC instructions in average, and large ratio of computation is utilized for CS_1. For instance, CS_1 of CBF utilizes 18.6% to 22.6% of CABAC instructions. SW design of CS_1 also reduces integratability of the encoder IP and increases bandwidth on system bus.

Scheme 4: Scheme 1 with HW Support of Residual SEs of BN and CS, Excluding CBF

In this scheme [95, 103], HW design of CABAC 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 in the total number of encoding bins, the remaining computation of SW is still significant (23.5% to 47.2% of CABAC instructions in CIF test, and 24.1% to 48.1% in HDTV 720p test in QP range of 12 to 36). Consequently, CABAC encoding can not be significantly accelerated, as the processing speed is restrained by the SW non-IP for the remaining part of CABAC functions. If the host processor can not generate and transfer

bin packets of non-residual SEs and CBFs as fast as the speed that bin packet are processed in the HW encoder, encoding procedure will pause and throughput is degraded.

(5) Scheme 5: Complete HW Function Support 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 hardware IP. In scheme 5, the functions SE encoding including binarization, CS, CA, and BAC are all partitioned to HW, and the remaining computation of SW is only data packet preparation for the HW encoder and coding result receiving from HW encoder. Compared to the other schemes of incomplete HW design of SE encoding, computation of host processor and data transfer on the system bus are minimized. Although design complexity of HW encoder is higher to support context selection in CS_1 and larger memory is required to store coded SEs that can be referenced by CS_1 during later MB coding, CABAC encoding is significantly accelerated because the bottleneck of SW non-IP is removed. Additionally, integratability of HW IP of CABAC encoder is also enhanced because it is easier to separate CABAC function from H.264 video encoding system to a function block. Considering the benefits of scheme 5, the proposed CABAC encoder IP of this thesis is based on this HW/SW functional partitioning scheme.

The proposed HW encoder design introduced in the following subsections is based on this selected HW/SW partitioning scheme (scheme 5) with completely support of SE encoding in HW. The key elements of this HW CABAC encoder include BN (binarization), CM (context modeling) in CS and CA (context access), and BAC (binary arithmetic coding). The elements of CS include CS_1 that needs to reference coded SEs of

neighboring MBs during context selection of current MB and simpler CS_2 with no reference of SEs of neighboring MBs. The functions of these key elements are further partitioned and reorganized properly in the encoder architecture during top-level HW functional partitioning to enhance average coding throughput and reduce clock cycle length, which will be discussed in the next subsection.

4.2.2 RDO Function Support in HW CABAC Encoder Design

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.

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%

The test results of Table 4-1 and Table 4-5 also indicates that compared to CAVLC in RDO-off mode, CABAC in RDO-on mode can achieve average bit rate reduction of around 20.4% in CIF coding and 29.1% in HDTV coding. It is beneficial to support both

RDO and CABAC, especially for high quality coding to obtain the significant benefit of the combinational coding gain of CABAC and RDO.

Computation of CABAC encoder significantly increases in RDO-on mode. As shown in the CIF sequence coding results of Table 4-6, the ratio of instructions of CABAC encoding to H.264/AVC of RDO-off mode is 0.17% to 1.18%, while the ratio is 6.64% to 20.63% in RDO-on mode. Support of RDO in HW CABAC encoder is necessary to accelerate CABAC encoding in RDO-on mode.

Table 4-6: Computation 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%

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 & restoration are required when RDO mode changes. The coding state of CABAC encoder includes three aspects: (a) the state of all context 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) coding rate accumulation and feedback of each RDO mode and (b) coding state backup and restoration in CABAC encoder. These two types of RDO operations are partitioned to HW IP in this thesis 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 Top-level HW Encoder Functional Partitioning

The key elements of HW encoder discussed in Scheme 5 of HW/SW functional partitioning in 4.2 include binarization in BN, context modeling in CM, and binary arithmetic coding in BAC. During HW top-level functional partitioning, functions of these 3 key elements are further partitioned and reorganized in the SE coding flow of HW encoder to maximize parallelism of different encoding elements and enhance coding speed of encoder. Two partitioning schemes are discussed, including (1) a preliminary scheme of FSM(Finite State Machine)-based architecture with low throughput and (2) the proposed scheme of parallel processing of SE encoding steps of BN, CS, and later full pipelined processing of bin packet generation (BS), context access (CA), and binary arithmetic coding (BAC). Both two schemes are introduced and compared to illustrate influence of functional partitioning to the encoder performance.

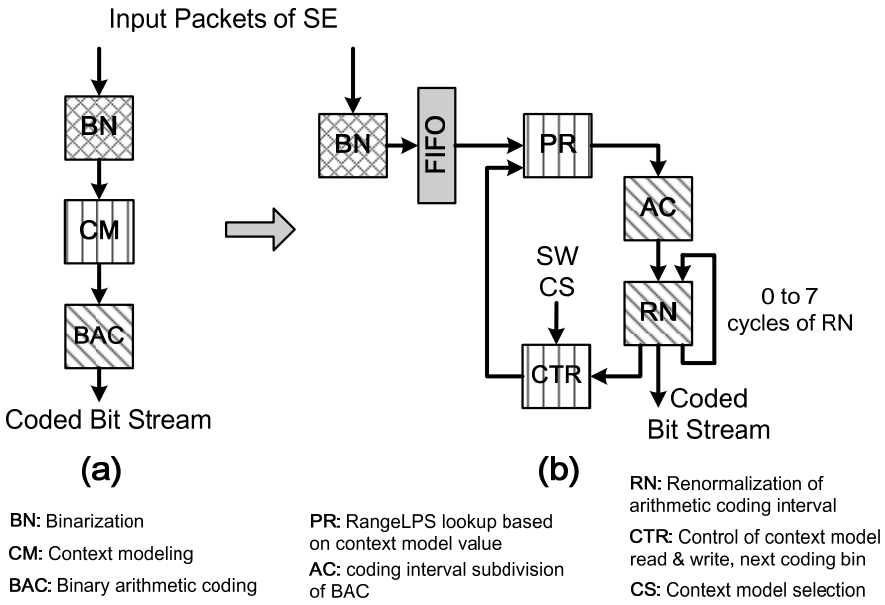


Figure 4-4: FSM-based HW CABAC encoder partitioning scheme.

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 shown in Figure 4-4 (a) can be implemented as FSM, which is shown in Figure 4-4 (b). 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, AC, RN, and CTR, 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 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 control state CTR, next encoding bin is 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 regular bin and 2 cycles for bypass bin). The throughput is not constant because renormalization of state RN, which is based on the reference algorithm of the standard [6], 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 of renormalization in RN.

4.3.1 Proposed Hardware Functional Partitioning Scheme

The analyses of HW/SW functional partitioning schemes aforementioned prove that it is necessary to support complete CABAC function of binarization, context modeling, and binary arithmetic coding in HW IP. FSM-based HW functional partitioning scheme is inefficient, because of sequential bin encoding manner. The proposed partitioning scheme is targeting at acceleration of CABAC encoding by exploiting parallelism of different bin encoding steps in maximum degree.

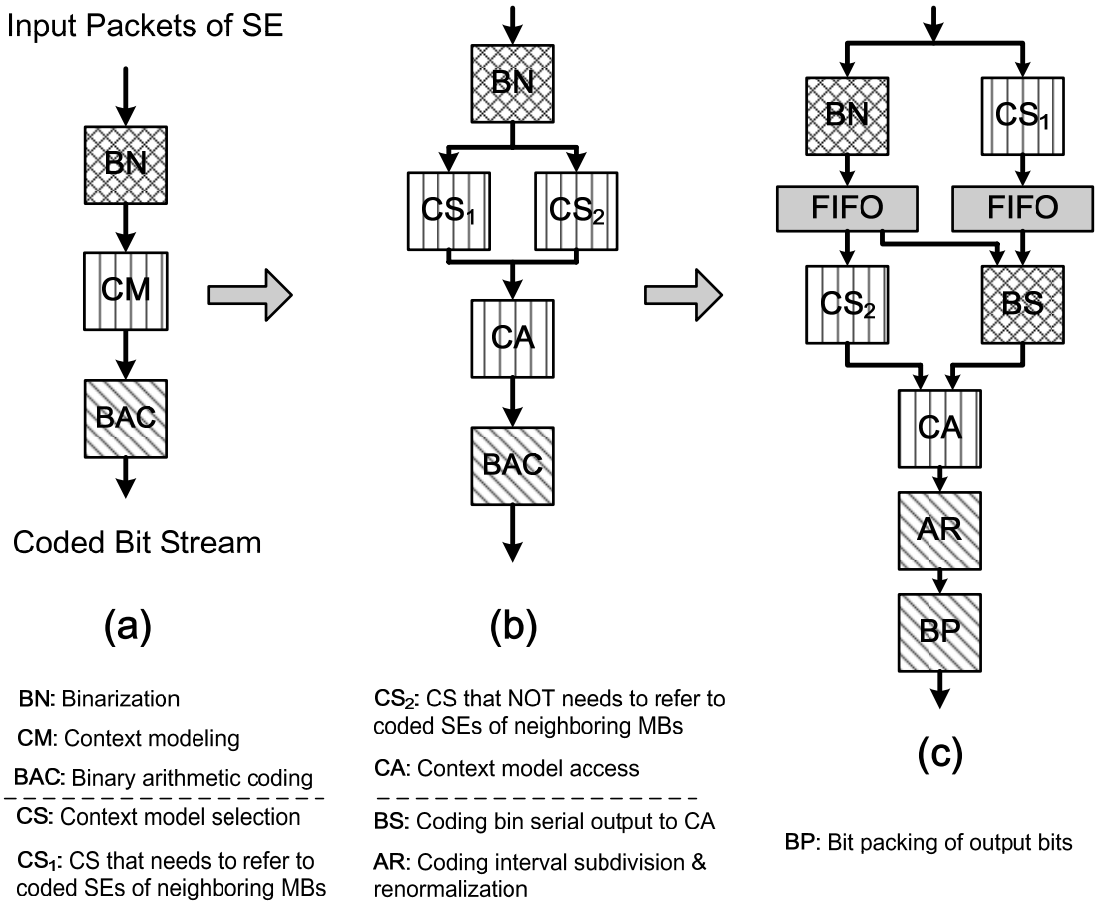


Figure 4-5: Proposed HW CABAC encoder partitioning scheme.

(1) Preliminary Partitioning

A preliminary stage of the proposed HW functional partitioning is shown in Figure 4-5 (b). At this stage, the three coding steps of CABAC algorithm shown in Figure 4-5 (a) 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. CABAC coding procedure is executed in sequential order of BN, CS, CA, and 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 scheduled after CS. After preliminary partitioning, further functional partitioning and adjustment are necessary targeting at high-throughput full-pipelined encoding. Data dependency between coding stages should be reduced, and computation complexity among partitioned function units needs to be balanced.

(2) Proposed Functional Partitioning Scheme for SE Coding

The proposed HW functional partitioning scheme for SE coding is illustrated in Figure 4-5 (c). Compared to the preliminary partitioning (Figure 4-5 (b)), unit BS is allocated to fetch bin string of SE generated by BN and serially generate packet of coding bins of bin string to CA & 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 critical path length of BAC, function of BAC is further partitioned to 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 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, critical path can be

efficiently reduced when the complex CtxIdxInc calculation is done 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 to 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.

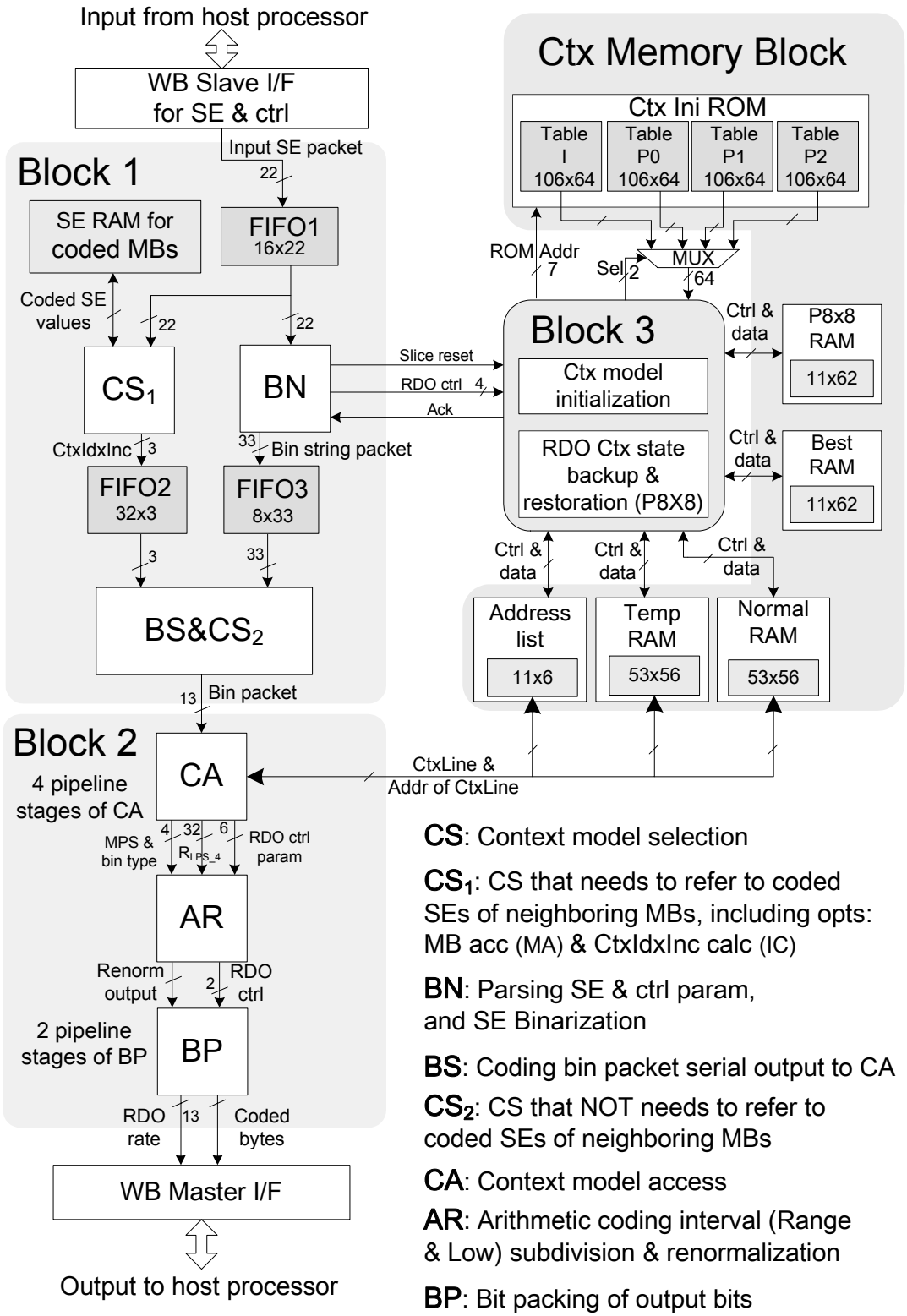
The function units partitioned in the proposed scheme (Figure 4-5 (c)) include BN & CS_1, BS & CS_2, CA, AR, and BP. The units are scheduled as sequential SE coding stages. Data dependency among these coding stages is minimized, and the units consist of a top-level SE encoding pipeline, in which all 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.

(3) Functional Partitioning of Additional Functions of CABAC Encoder

As discussed in 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 thesis.

4.3.2 Full-Pipelined Top-level HW CABAC Encoder Architecture



CS: Context model selection

CS_1: CS that needs to refer to coded SEs of neighboring MBs, including opts: MB acc (MA) & CtxIdxInc calc (IC)

BN: Parsing SE & ctrl param, and SE Binarization

BS: Coding bin packet serial output to CA

CS_2: CS that NOT needs to refer to coded SEs of neighboring MBs

CA: Context model access

AR: Arithmetic coding interval (Range & Low) subdivision & renormalization

BP: Bit packing of output bits

Figure 4-6: Block diagram of top-level architecture of HW CABAC encoder.

The top-level architecture of the proposed CABAC encoder is shown in Figure 4-6. The encoder consists of three functional blocks and a memory block: Block 1 for input parameter parsing & binarization and context model selection (CS); Block 2 for context model access (CA) and binary arithmetic coding (BAC); Block 3 for functions of context model initialization and context state backup & 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 function units CS_1, BN, BS&CS_2, 3 FIFO buffers, and a RAM of coded SEs that is only accessed by CS_1. Unit BS and CS_2 are integrated into one unit in order to complete operations of context model selection (CtxIdx calculation) and packing of bin and CtxIdx in the same cycle. Block 2 consists of 3 function units including CA, AR, and BP, as introduced in 4.3.1. The WISHBONE (WB) system bus [104] master and slave interfaces 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 Chapter 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, while the packet read from FIFO1 is parsed in BN and CS_1 simultaneously. Context models of part of regular bins are selected in CS_1 by calculating a 3-bit CtxIdxInc when coded SEs of neighboring blocks/MBs need 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, 13-bit bin packet containing bin value, bin type, and CtxIdx of regular bin is generated in S&CS_2 and sent to Block 2. As previously discussed, FIFO2 (32-word×3-bit) and FIFO3 (8-word×33-bit) are inserted in Block 1 to buffer the bin string of BN and CtxIdxInc of CS_1 that can largely reduce pause probability of unit BS&CS_2 and following coding stages when the input data of BS&CS_2 is not ready. In Block 2, context model of each regular bin is accessed in unit CA from context RAMs according to CtxIdx. Coding interval is subdivided and renormalized in unit AR by updating values of Range & 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 Figure 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 Chapter 5.

4.3.3 Date Dependency Removing & Encoding Acceleration

Data dependency of CABAC encoding algorithm can not be reduced efficiently in SW implementation. In the reported HW designs, data dependency needs to be efficiently removed to enhance coding throughput and clock frequency. Data dependency of CABAC encoding algorithm and proposed techniques of data dependency removing and encoding acceleration are briefly discussed here:

(a) During bin coding, context models of regular bin need to be accessed from context RAM, updated based on bin value, and stored to RAM. Same context model can be accessed by successive coding bins, and RAM access delay will cause bin coding pipeline pause. The solution of this design includes techniques of access context model by context line and local buffer context models. Updated context models can be accessed in the next cycle from local buffer instead of context RAM. Context models are also prefetched according to context selection results of two CS units in pipeline manner before they are used.

(b) For each bin coding, coding interval update and renormalization, and bit packing and bit stream output consist of sequential operations of complex computation. Range of LPS of next coding bin can only be determined after update and renormalization of coding interval of current bin. Because of data dependency, direct implementation of bin coding results in long critical path length and low coding speed. To remove data dependency of bin coding, all possible $Range_{LPS}$ values are prefetched in unit CA based on the selected context model. In unit AR, corresponding $Range_{LPS}$ is selected when the updated Range is available. Following computation of parsing, packing, and output of coded bits have no feed back to the coding interval update of AR, and this

part of computation is assigned to later pipeline stages to reduce critical path length of Block 2.

(c) In SW implementation, bin generation of SE binarization and context selection of bin are processed sequentially for each binarized bin. It is because context selection of current bin can reference previous bins of same SE or coded SEs of neighboring coded block or MBs. Block 1 design of the proposed HW encoder removes the dependency by dividing binarization and context selection into two sequential units of BN and CS_2 with FIFO insertion to enable parallel processing of the two steps at higher speed. The more complex context selection of CS_1 is precalculated in parallel with BN to minimize critical path length in unit CS_2. Coded SEs of neighboring MBs are prefetched and local buffered during current MB coding in CS_1, so that data dependency of reference SEs is removed during context selection.

(d) Context selection and context access are two sequential coding steps with no feedback loop. Instead of sequential computing of CS and CA in SW algorithm, the two steps are divided into different units with pipeline buffer insertion. Therefore, CS and CA can process data in parallel.

The techniques introduced above are applied in the proposed CABAC encoder architecture with efficient removing of data dependency and acceleration of CABAC encoding. Parallel processing of binarization, context selection, and bin coding are achieved in the encoder.

4.4 Binarization and Generation of Bin Packet

4.4.1 Input SE Parsing & Binarization of Unit BN

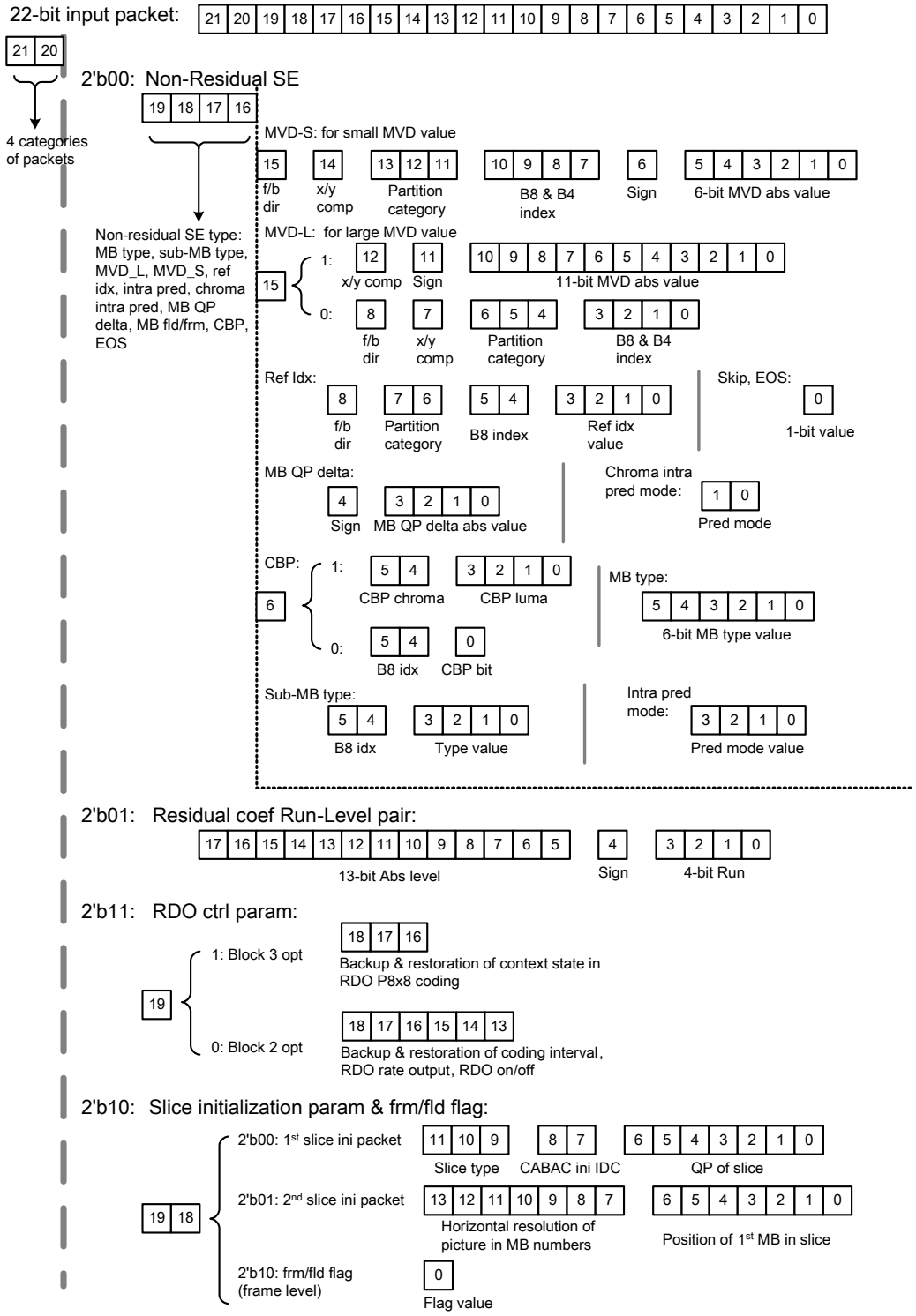


Figure 4-7: Input packet format of CABAC encoder.

Input packets of CABAC encoder provide necessary data for both SE coding and encoder control. Input packets are buffered in FIFO1 and accessed by both unit BN and CS_1. The format of the 22-bit packet is illustrated in Figure 4-7. The top 2 bits (bit 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 frame level. For non-residual SEs, 4-bit tag (bit 19~16) is used to classify different SE types including MVD-S (MVD abs value that can be represented in 6 bits), MVD-L (for large MVD value), reference index, MB QP delta, Skip flag, 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 13-bit absolute value of level, and 4-bit run, which are the maximum range of Main profile. Two types of RDO control parameters are supported that control RDO operation of context state backup & restoration in Block 3 and coding interval backup & restoration and coding rate output of each RDO mode in Block 2. Slice initialization parameters are parsed from two input packets including slice type, slice QP, CABAC initialization IDC, number of MBs in the horizontal resolution of frame/filed, and position of the 1^{st} 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 is illustrated in Figure 4-8. BN contains 3 major data paths that process input packets of non-residual SEs, residual SEs, and encoding control parameters. When control parameters of Block 3 are parsed and sent to Block 3, BN waits for acknowledgment of Block 3 before processing next input packet. End-of-slice (EOS) flag and RDO operation parameters including

Range and Low backup & restoration modes, RDO on/off flag, and RDO coding rate output instruction are buffered in FIFO3 and forwarded to units of Block 2.

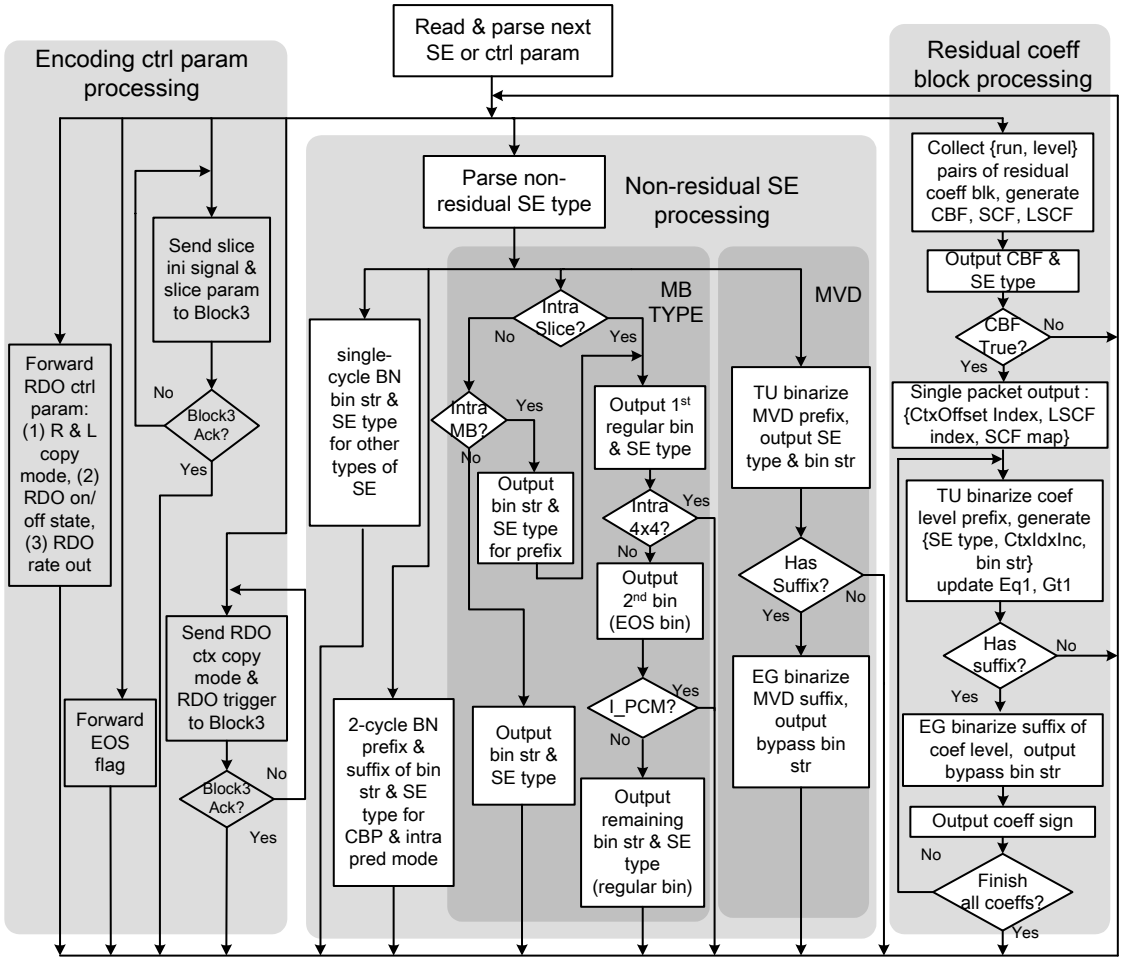


Figure 4-8: Procedure for parsing and binarization non-/residual SE and control parameters of unit BN, Block 1.

Different types of non-residual SEs shown in Figure 4-7 are parsed according to the 4-bit tag in the packet and binarized in BN. Except SE type and SE value that are utilized for binarization, addition 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 direction, partition category, and block index of 8×8 sub-MB and 4×4 block which provide 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 Figure 4-7. Throughput of CABAC encoder is not influenced by such packet partitioning, because 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. Barrel shifter is utilized to implement binarization schemes of unary or 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 slices, LUT is implemented by combinational circuits instead of memory-based LUT to reduce table lookup delay.

Coefficients of residual block are processed in a different path, 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 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 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 1st 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 Figure 4-8. The counters of Eq1 and Gt1 are accumulated for each block during coefficient level binarization procedure.

```
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]
```

Figure 4-9: HW-oriented EGk binarization algorithm.

For SEs of MVD (motion vector difference) and residual coefficient levels, binarization methods of TU (truncated unary) and EGk (kth order Exp-Golomb code) are used to binarize prefix and suffix bin string, respectively. SW-based implementation of EGk binarization introduced in Chapter 2 is not suitable for HW implementation because of long and variable operation 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). In general, the HW-oriented EGk binarization algorithm can be described in the following pseudo code, shown in Figure 4-9.

According to the algorithm of Figure 4-9, EG3 circuits that support 11-bit range of absolute MVD is designed to generate MVD suffix bin string in single cycle, as shown in Figure 4-10(a). 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 Figure 4-10(b).

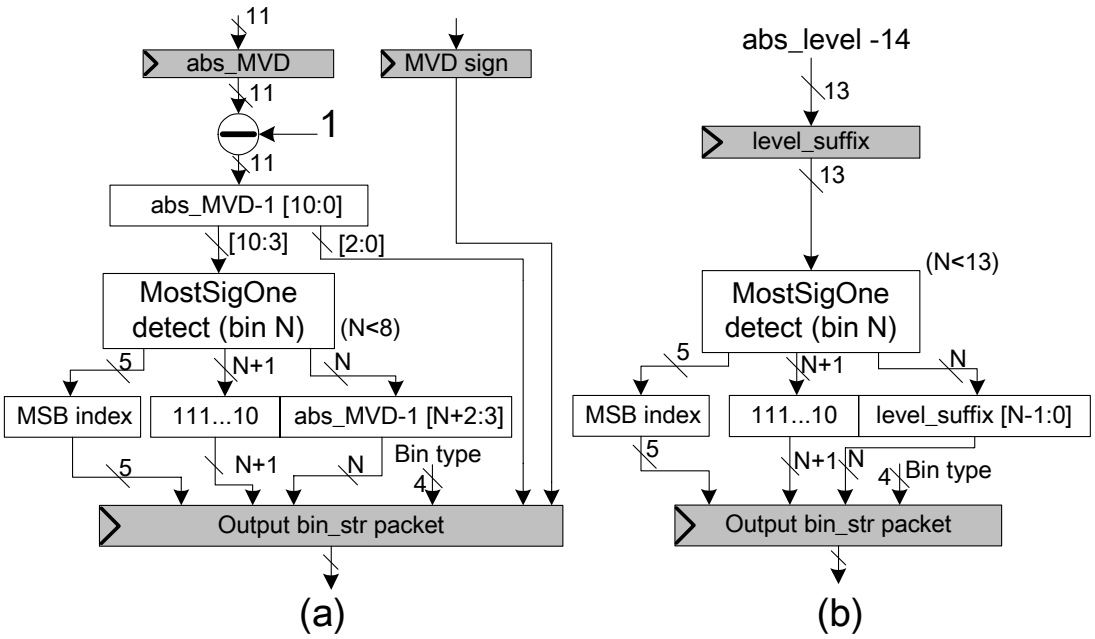


Figure 4-10: Fast EGK binarization implementaion. (a) EG3 binarization for the suffix of MVD; (b) EG0 binarization for the suffix of abs_level_minus1.

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 Figure 4-11.

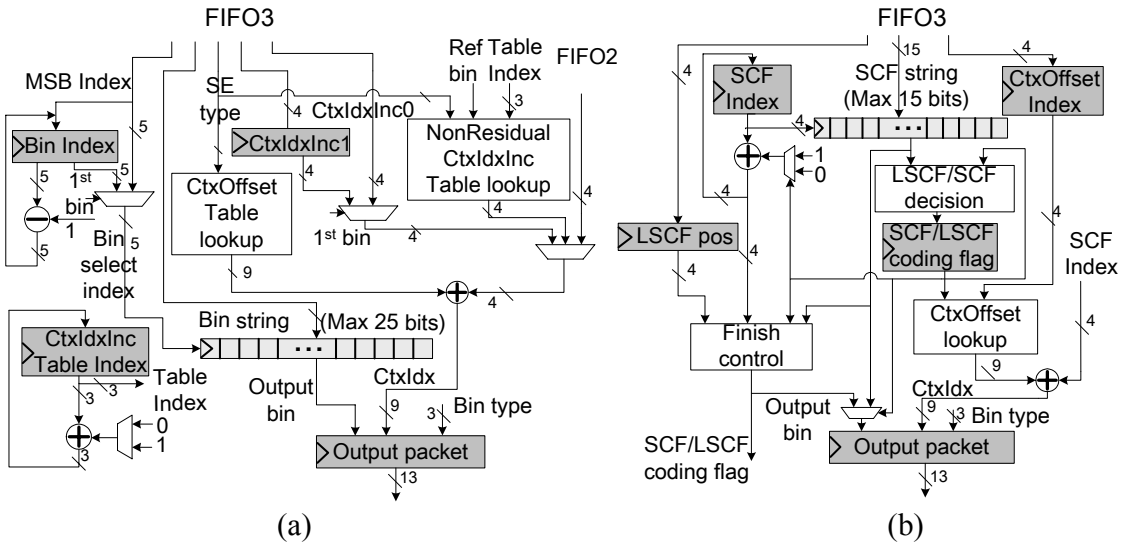


Figure 4-11: Architecture of unit BS&CS2: (a) CtxIdx calculation and bin packet serial output circuit for all SE, excluding SCF and LSCF; (b) CtxIdx calculation and SE serial output of SCF and LSCF packet of residual coefficient block.

Figure 4-11(a) 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 Figure 4-11(b). 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 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 Chapter 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 & 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 & output of bin packet of next bin string with no pause. Therefore, constant throughput of 1 bin packet per 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 subsections, 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 Proposed Renormalization & 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 proposed in this thesis, which provides identical results, but significantly higher coding throughput, compared to the reference algorithm of H.264/AVC [6]. The proposed algorithm is different from the QM coder renormalization derived in [93, 95]. After coding interval subdivision, Range & Low of selected subinterval are input to the renormalization & bit packing stage. The complexity of this algorithm is in two aspects: leading zeros detection (LZD) of updated Range of coding interval to determine shift length of renormalization, and least significant zero (LSZ) detection of bit string shifted out from Low of interval to determine output bits and outstanding bits. Fast

combinational circuits can be designed to accelerate LZD and LSZ detection, instead of recursive renormalization and bit output procedure of SW implementation.

The proposed algorithm can be partitioned into two pipeline stages 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 divided into two pipeline stages, 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 [93, 95].

The proposed renormalization and bit packing algorithm is implemented as a 3-stage pipeline illustrated in Figure 4-12. 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 & 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.

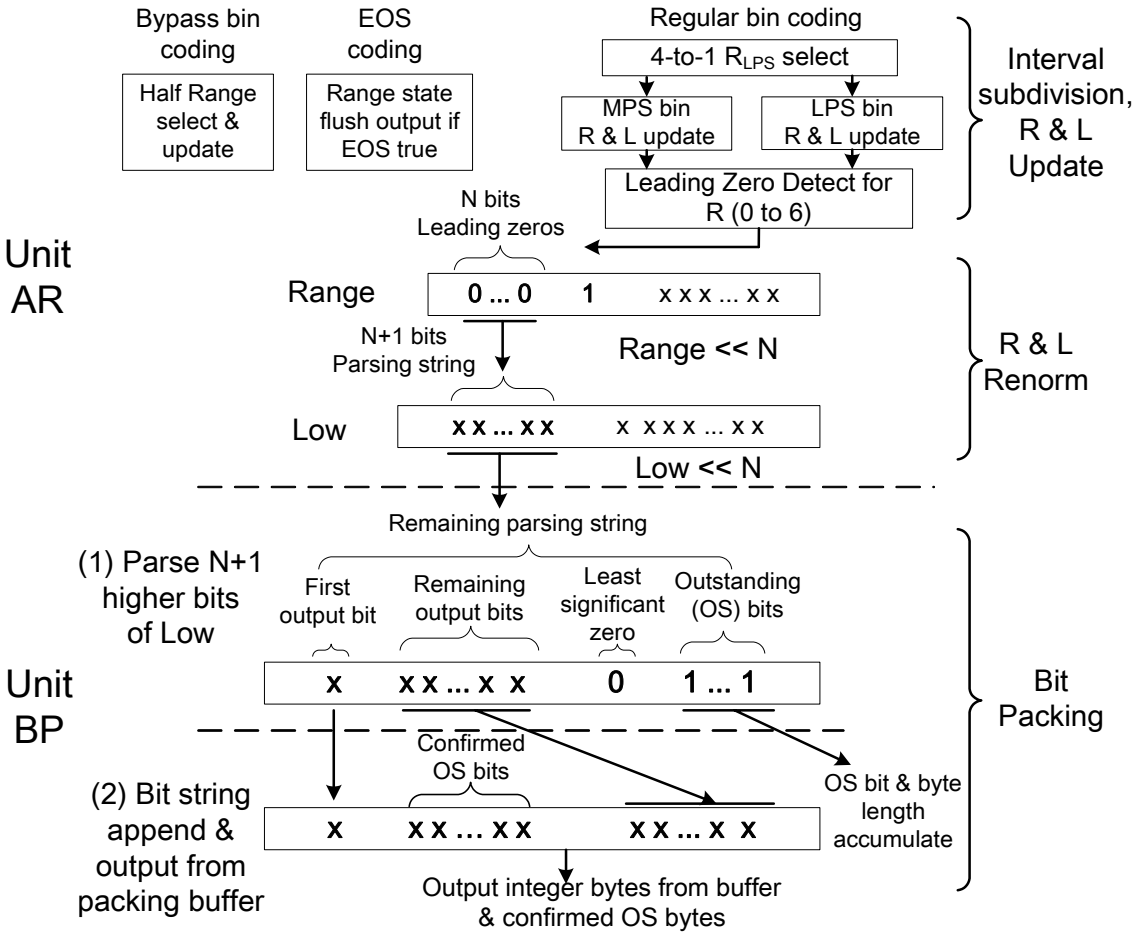


Figure 4-12: Three-stage pipeline implementation of renormalization and bit packing algorithm in unit AR and unit BP.

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 Figure 4-12, 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 & Renormalization of Unit AR

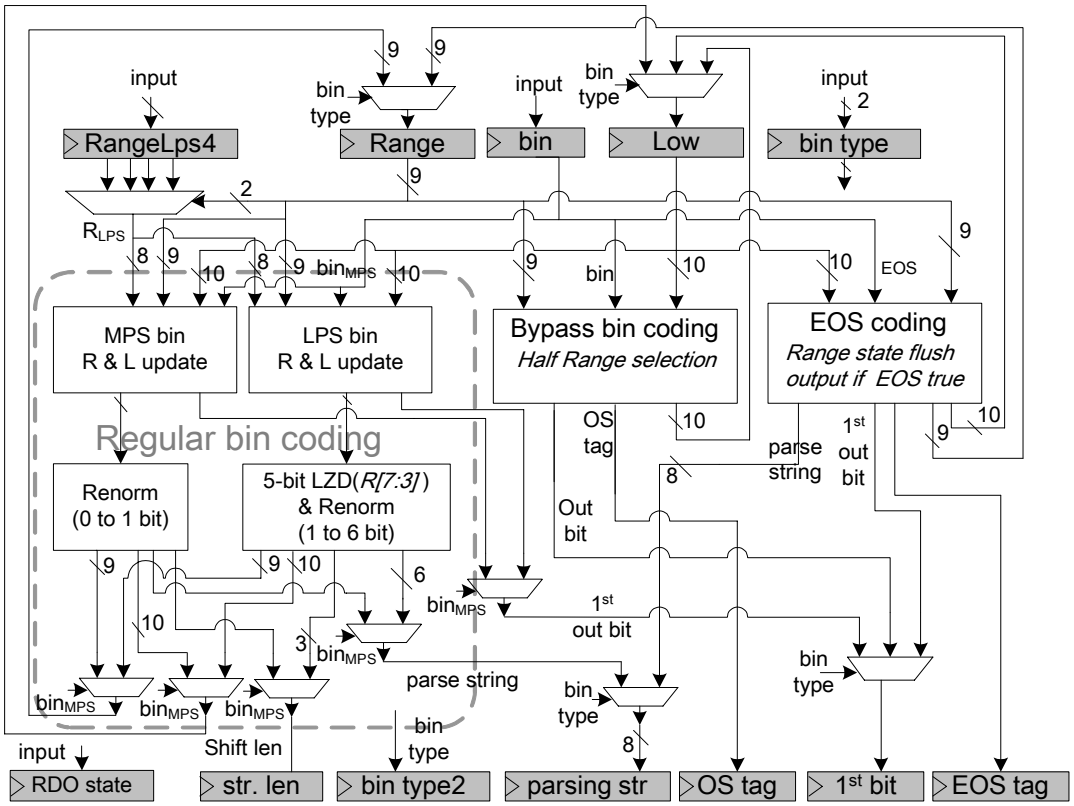


Figure 4-13: Architecture of unit AR.

The architecture of unit AR is shown in Figure 4-13. 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 Figure 4-13. For MPS bin, renormalization is not needed or only 1 bit renormalization is taken; while for LPS bin, length (1 to 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 & 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 $Range_{LPS}$ selection to coding interval update and renormalization of LPS bin, to the data input of Range registers. In [95], update and renormalization of Low is scheduled one cycle later than that of Range, because selection of $Range_{LPS}$ 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 Figure 4-14. 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 bits 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 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.

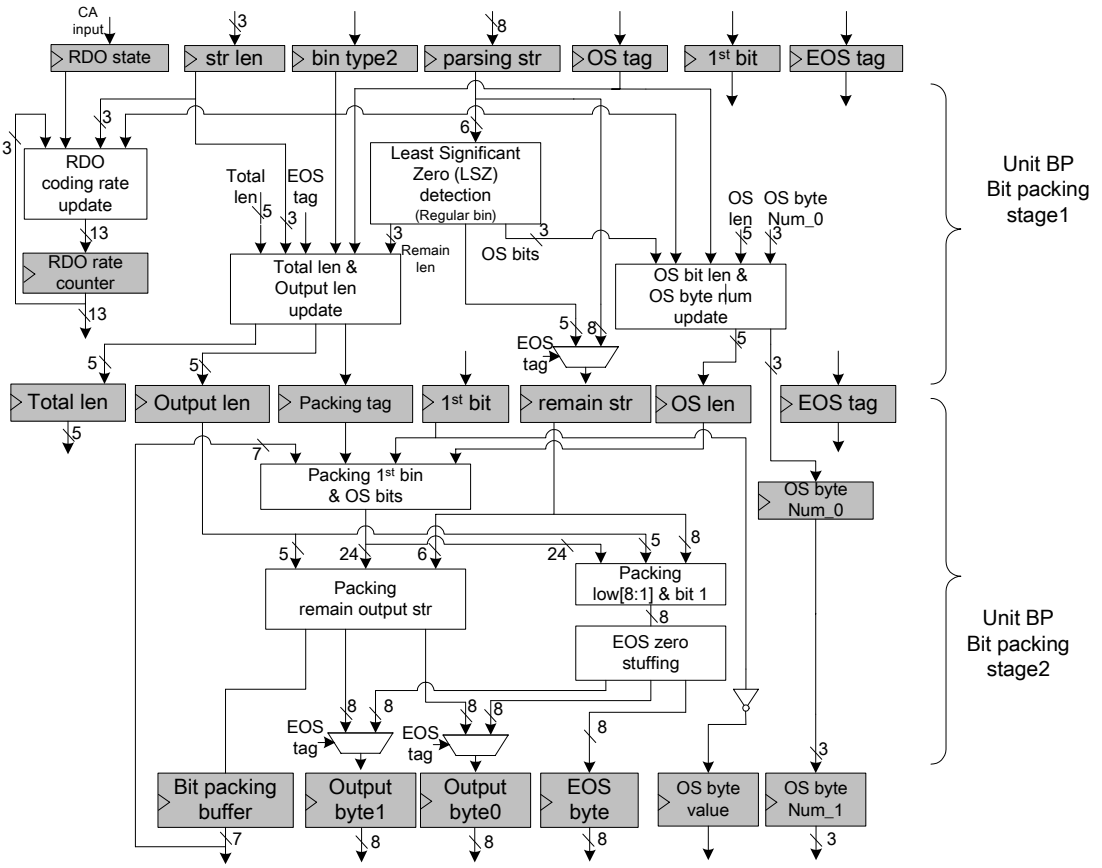


Figure 4-14: Two-stage design of bit packing.

As shown in Figure 4-14, 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 CABAC encoder designs [94, 95, 105] 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 of 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 [6]. Compared to most of the reported designs, context model initialization is implemented in HW in the proposed encoder. The advantage of HW 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 Figure 4-6, one of 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. Total of 424 context models can be initialized in 110 cycles, including pipeline preparing time.

4.6.2 RDO Function Support in BAC

During RDO-on coding, according to the RDO control parameters, state of coding interval (values of Range & Low) are backed up or restored in unit AR when RDO mode changes. Three backup copies of Range & Low values are allocated. During 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 (um 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 aiming for 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.

Chapter 5 Efficient Architecture of CABAC Context Modeling

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 & local buffering is proposed to reduce RAM access frequency. Coding state backup & 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 & restoration of RDO-on mode is only proposed in this thesis.

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 CABAC. In order to select most suitable context model of each regular bin, a number of factors need to be analyzed, including SE type, coded bins of current SE, coded SEs of neighboring blocks, etc.

As introduced in Chapter 2, context model of context RAM addressed by CtxIdx, which is the sum of CtxOffset and CtxIdxInc. Function of context model selection is partitioned to unit CS_1 and unit BS&CS_2 of Block 1. In CS_1, context model selection needs to reference to coded SEs of neighboring blocks of current MB or neighboring MBs. As shown in Figure 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 of backup coded SE. Generated CtxIdxInc values of CS_1 are buffered in FIFO2 and utilized by BS&CS_1 for CtxIdx calculation. In the following subsections of 5.1, the scheme of storage and fast access of coded SE in IC sub-unit will be introduced first, followed by separate discussion of the architectures of IC and MA sub-units.

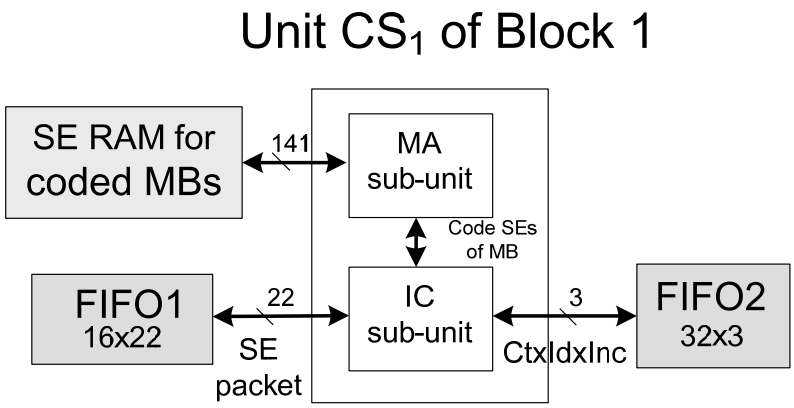


Figure 5-1: Block diagram of unit CS_1, including MA sub-unit and IC sub-unit.

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

IC sub-unit of unit CS_1 calculates CtxIdxInc value for particular bins at specified position (bin index) of several types of SEs, as specified in Table 9-30 of [6]. Coded SEs of neighboring BPMB (4×4 block, MB/sub-MB partition, or MB) on the top or left of current BPMB are used for the CtxIdxInc calculation of one regular bin of current SE in IC sub-unit.

Implementation of CtxIdxInc calculation with reference to the coded SEs in unit CS_1 is challenging because the irregular manner of accessing 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 [6] can be described as the following steps: (a) for the processed bin, the category of the SE is identified first whether the SE is of MB, partition, or block; (b) 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; (c) the SE of neighboring block, partition, or MB is accessed from memory and used for CtxIdxInc calculation. The SW-oriented BPMB derivation is complicated and not suitable for HW design targeting at fast access and high throughput. In order to complete derivation of neighboring BPMB, access of coded SE of the BPMB, and 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.

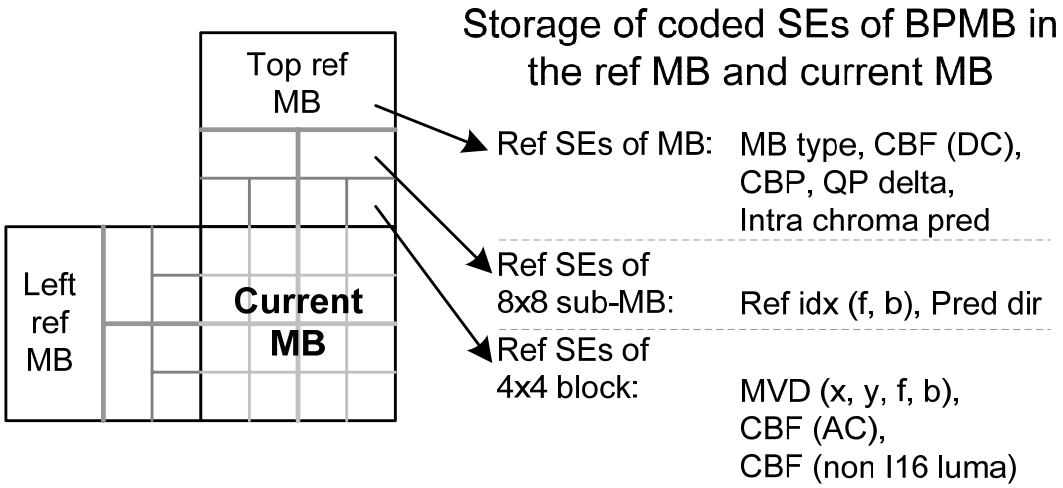


Figure 5-2: Reference MBs on the top and left of current MB, and storage of 3 categories of coded SEs (MB, 8×8 sub-MB, and 4×4 block) in the reference BPMB of current and reference MBs.

As shown in Figure 5-2, storage of coded SEs of variable types of BPMB are classified to 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 partition 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 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 of coded SE is significantly more efficient in circuit complexity and processing speed (throughput × clock frequency) and for HW design, compared to SW-oriented procedure aforementioned.

(1) Fast Access of Coded SEs of Neighboring Block or sub-MB

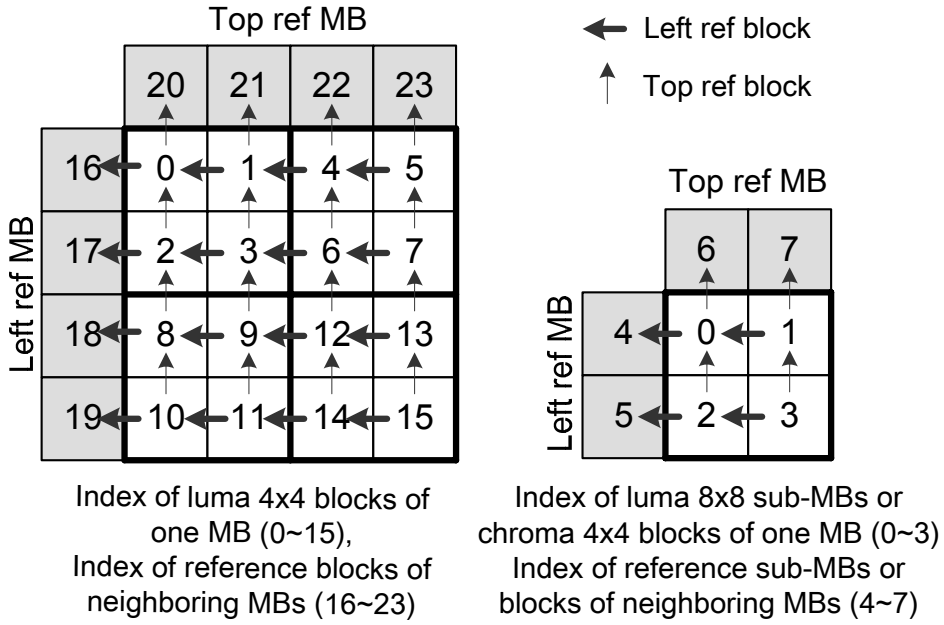


Figure 5-3: Fast access of neighboring coded block and sub-MBs. (a) Access of neighboring luma 4×4 blocks, and (b) access of neighboring 8×8 sub-MBs and chroma 4×4 blocks of 4:2:0 video format.

Fast access scheme of neighboring coded block or sub-MB of IC sub-unit is illustrated in Figure 5-3. For block level SE, a four-bit block index of the SE is provided in input SE packet. The range of block index is from 0 to 15, with higher 2 bits of index locating one of four 8×8 sub-MB in the MB and lower 2 bits locating one of 4 blocks in the sub-MB. The 4 blocks of left reference MB are indexed from 16 to 19, and the 4 blocks of top reference MB are indexed from 20 to 23. The derivation of block index of left and top block of current block is based on combinational circuit implementation of table lookup.

The lookup table (LUT) of block index can be derived from Figure 5-3 (a), 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.

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

The 8×8 sub-MBs and chroma 4×4 blocks of 4:2:0 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 Figure 5-3 (b). The two sub-MBs or blocks of the left reference MB of current MB are indexed with value 4 and 5, while the two of top reference MB are indexed with value 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 2-bit index of current MB. The LUT can be derived from Figure 5-3 (b), and it is implemented as 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:

- 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; else, the sub-MB index is looked up in the LUT (Table 5-2) based on 2 higher bits of block index of current block

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

To accelerate access of coded SE in this situation, a fast LUT is established using the two-step derivation, and it is 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 LUT input 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 Figure 5-3 (a) and (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 of LUT and direct read of SE value from local registers, instead of complicated calculation and memory access of SW-oriented procedure.

(2) Storage of Coded SEs of Reference MB

The storage of codes SEs of the top and left reference MBs are shown in Table 5-4. Total of 141 bits are allocated for top or left reference 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 0 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 size of SE RAM reduces 31.2%.

Table 5-4: Storage of coded SEs of top/left reference 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 slice	
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 per MB, indicating if QP delta is 0 or not			

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 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 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 Figure 4-7, additional 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 Figure 5-4, including CtxIdxInc calculation of several types of non-residual SE, CtxIdxInc calculation of

residual block (CBF), backup & 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 [90], in which 4 cycles are required to access codes SEs of top and left reference BPMBs and 1 more cycle to select context model. Compared to [90], 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 constant high throughput of CABAC encoding. Functions of CtxIdxInc calculation and storage of coded SE values of different SE types are discussed separately.

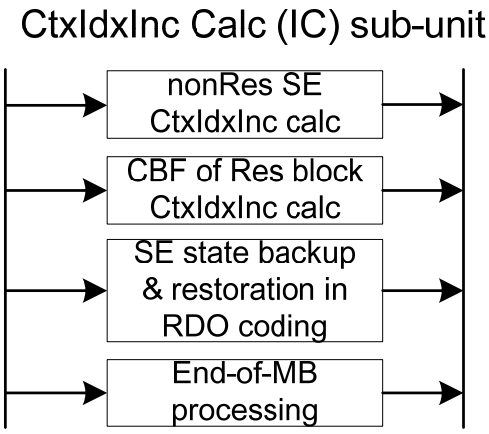


Figure 5-4: Functions of IC sub-unit of unit CS_1.

(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.

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)

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 & B4 index) from input packets, as shown in Figure 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.

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.

(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 subsection.

Table 5-6: Classification of MB type and stored values of MB type

MB type	3-bit Stored Value	Descriptions
Intra4×4 (I4)	0	Intra-coded MB in I, P, and B slices
Intra16×16 (I16)	1	
I_PCM	2	
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

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.

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.

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			

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).

(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 [82]. During processing of current MB in [82], 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 [82], the situation is avoided in the proposed encoder design.

5.1.3 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 frame/field, 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 frame/field 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 & 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 Figure 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 sub-unit is idle, while IC sub-unit continues processing following packets of MB_i till the end flag of MB_i.

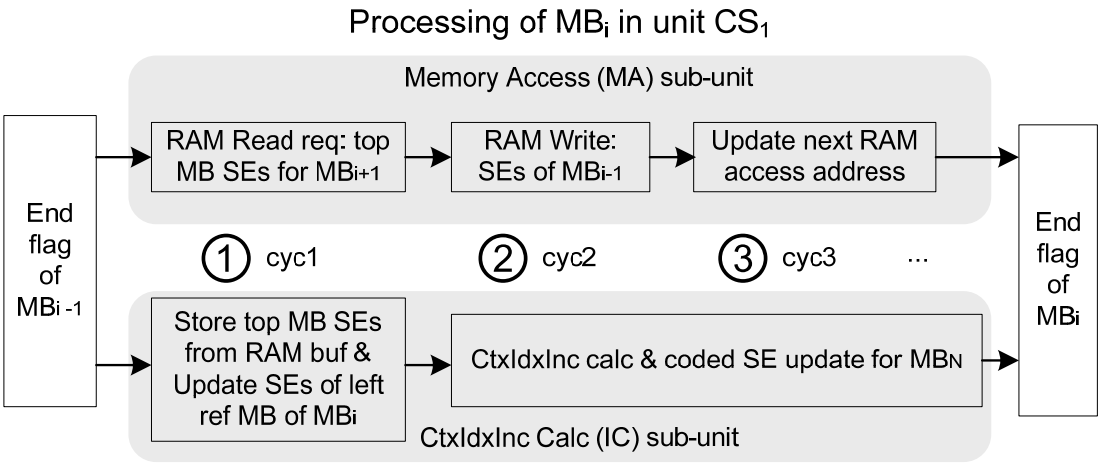


Figure 5-5: MB processing in MA sub-unit and IC sub-unit of unit CS_1.

One example of 3-cycle MB processing scheme of MA sub-unit is shown in Figure 5-6. N is MB row number. M is number of MBs in the horizontal direction of coding frame/field (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 $MB_{N,0}$. 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.

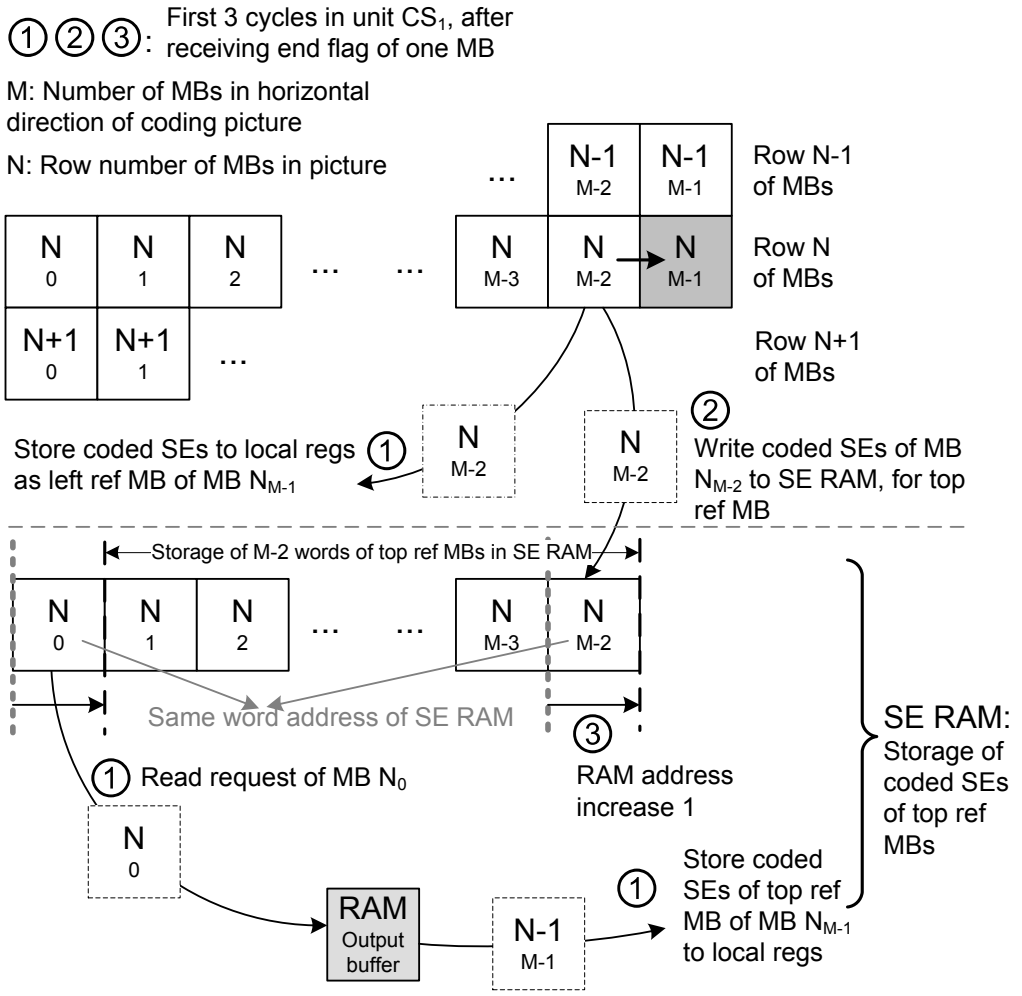


Figure 5-6: Operations of MA sub-unit in the first 3 cycles of $MB_{N,M-1}$ processing.

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 [82, 90] that support context model selection of unit CS_1, because the RAM is read and write only once per MB; (b) compared to [82], RAM size is reduced with the proposed RAM read & 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 [90]; (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 [86, 89, 90, 93, 105, 106], 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 [87]. 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 & Local Buffering

In this thesis, an efficient context model access scheme is proposed and implemented in unit CA, as shown in Figure 5-7. Before design of unit CA, computational and data transfer complexity analyses have been performed on a reference CABAC code [107], 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 [86, 89, 90, 93, 105, 106], the proposed CABAC encoder accesses the context models of context RAM by context lines, each of which contains eight 7-bit 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 [87, 95] 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.

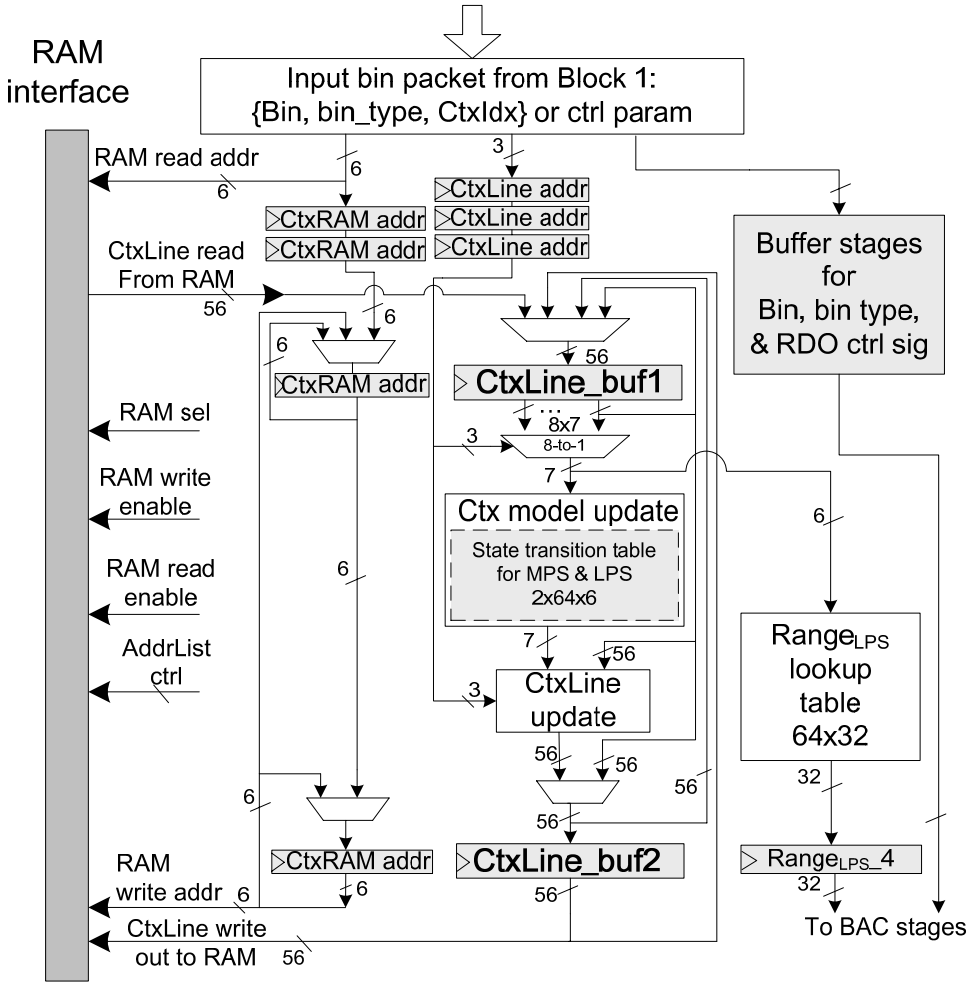


Figure 5-7: Architecture of unit CA with pipelined context line access and local buffering scheme.

In Figure 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 & 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 & 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

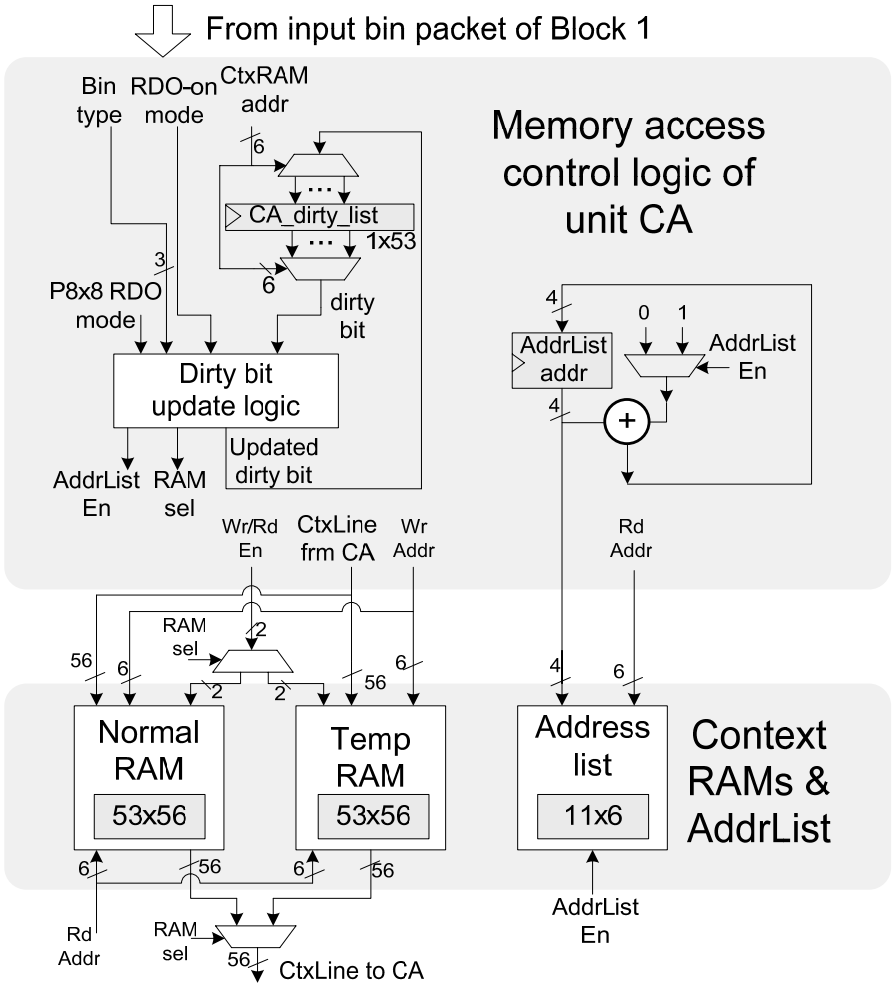


Figure 5-8: Architecture of memory access control of unit CA in both RDO-off and RDO-on mode.

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 [93]. In this thesis, 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 Figure 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 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 & 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 Figure 5-9. In the cache-based designs [87], 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 [6]. 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. 53 context lines (424 context models) are allocated in the context RAM with only 6.8% dummy context models inserted.

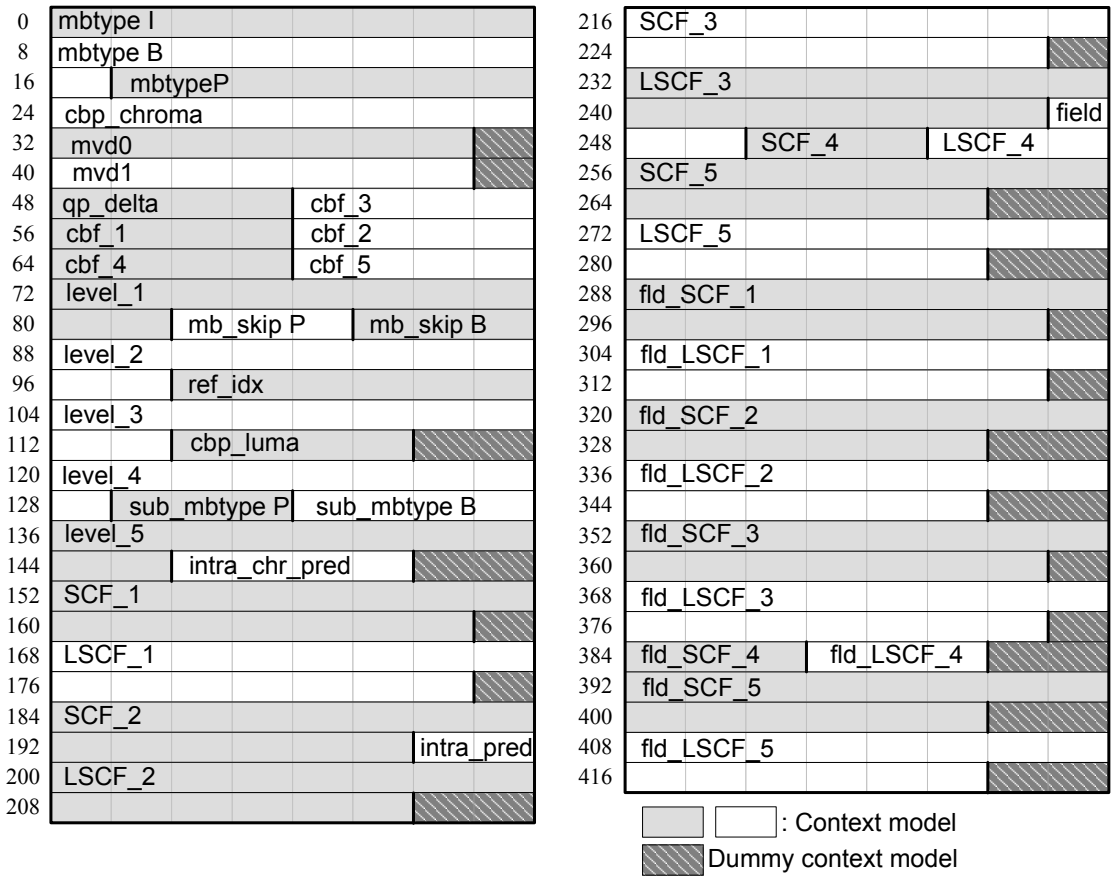


Figure 5-9: Reallocation of context model in context RAM (Normal RAM). Context models of Normal RAM are illustrated as two continuous parts in the figure.

5.3 Context State Backup & 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 [93].

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 & 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 & restoration.

As shown in Figure 5-10, four pipeline architectures are designed in Block 3 to support 4 types of context state backup & restoration operations during P8×8 RDO coding with data transfer speed of 1 context line per cycle. Buffering stages of 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.

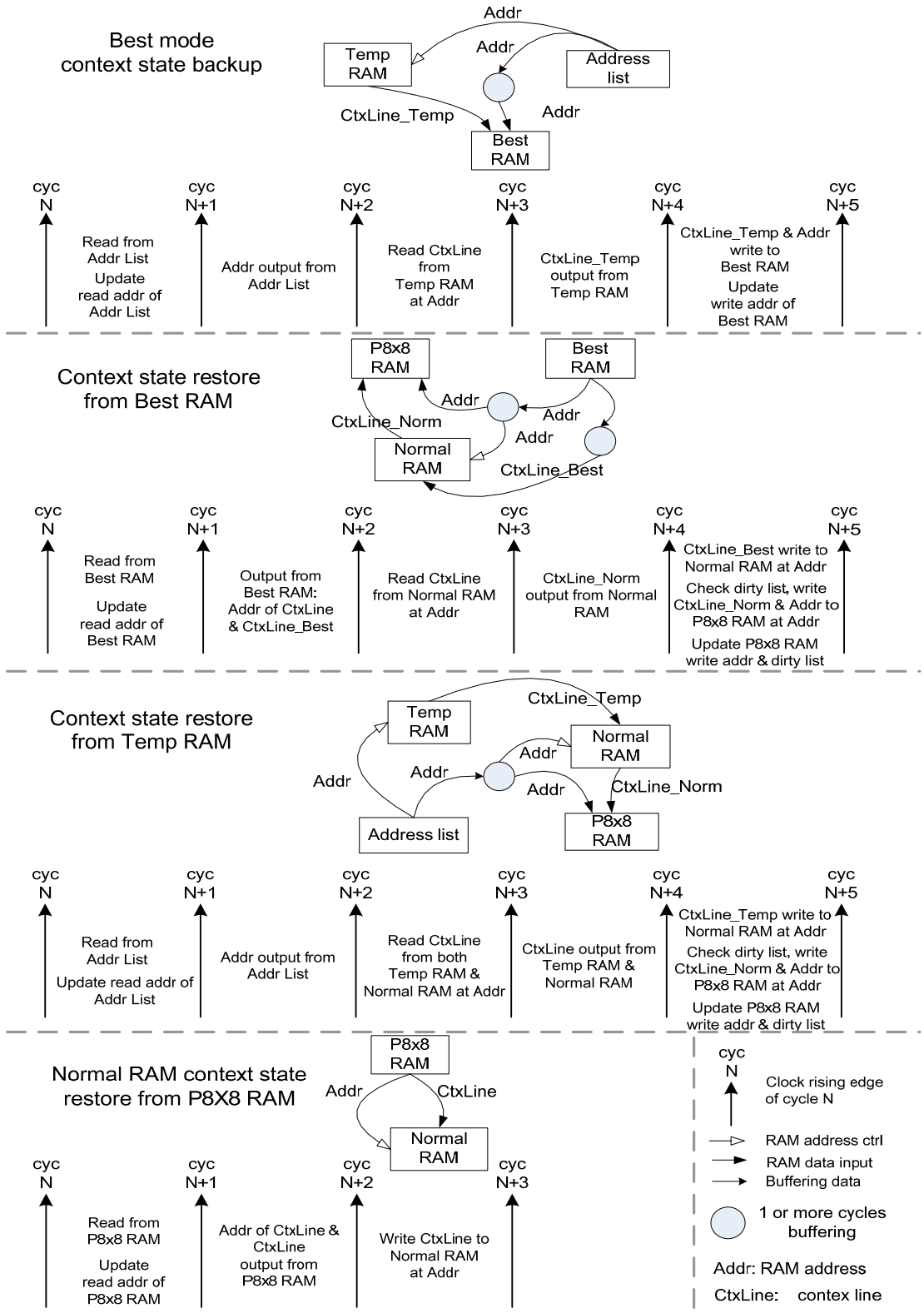


Figure 5-10: Four types of pipelined context state backup & restoration operation in P8×8 RDO coding.

- *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 & restoration and context line access scheme of context RAM, the operational delay of context state backup & 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 Chapter 7.

5.4 Coded SE State Backup & 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.

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

Four types of coded SE state backup & restoration operations are supported, and can be triggered by the same control parameters that also control operations of context state backup & 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 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 of CABAC encoder is proposed for the context model selection and context model access in both RDO-on and RDO-off modes. For the function of context model selection, single-cycle CtxIdxInc calculation of regular bin is achieved for all types of SEs, which can be attributed to the proposed efficient scheme of storage & fast access of coded SEs of current MB and reference MBs. Additionally, RAM access of coded SEs is significantly lower, compared to the reported designs [82, 90]. For the function of context model access, a pipelined context line access & local buffering scheme is proposed, which can be significantly

reduce context RAM access frequency. In order to fully support RDO in CABAC encoder, operations of backup & restoration of context state and coded SE state are implemented in the 3 functional blocks of CABAC encoder. Coded SE state backup & restoration is only supported in the proposed design, and context state backup & restoration is more efficient compared to [93], with significant lower operation delay and smaller size of context RAMs.

Chapter 6 System Bus Interface and Inter-connection Design

As introduced in Chapter 4, one step of 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 [104] system bus of CABAC encoder is discussed after introduction of WISHBONE system bus specification. Crossbar system bus inter-connection is also designed, which enables convenient integration of the proposed CABAC encoder with other function components in the H.264/AVC encoding system.

6.1 Introduction of the WISHBONE System Bus Specification

WISHBONE system bus provides a flexible design methodology to create a common interface between 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 does not constrained the creativity of core developers or end users.

6.1.1 Interface Signals of the WISHBONE System Bus

The simplest point-to-point interconnection of the WISHBONE bus that directly connects one master and one slave is illustrated in Figure 6-1.

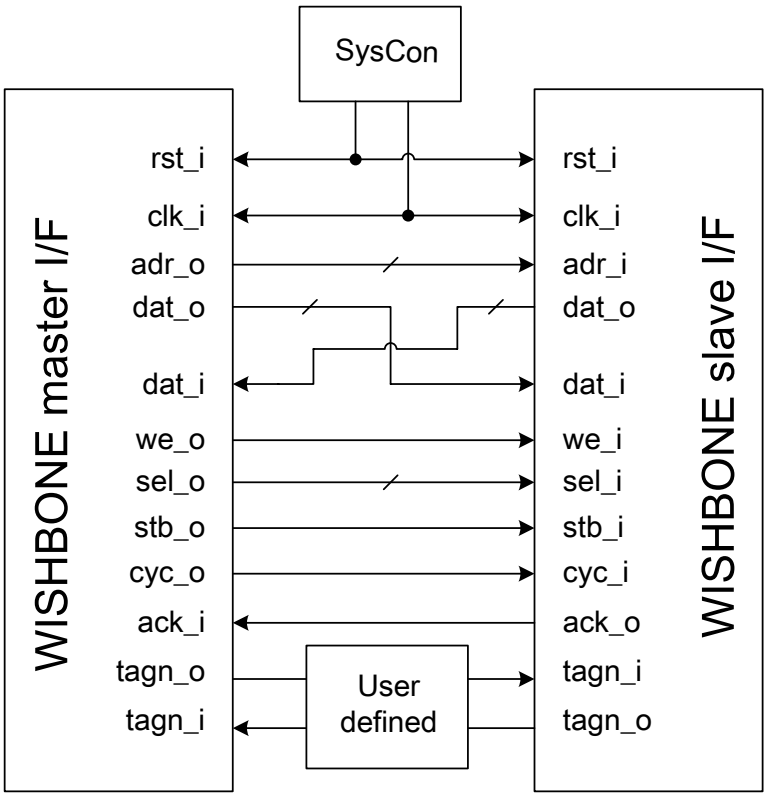


Figure 6-1: Point-to-point inter-connection of single master & slave of the WISHBONE system bus.

Input signal of interface is named with System reset (rst_i) and clock (clk_i) signals are generated by 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.

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 & err_i	Cycle termination signal from slave indicating retry & error of the data transfer; the signals help to enhance robustness of data communication	rty_o & err_o
tagn_i & tagn_o	Tag values that provide additional information of data bus, address bus, and bus cycles	tagn_o & 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. Bus cycle is defined [104] as the procedure whereby digital signals affect the transfer of data across a bus by means of an interlocked sequence of control signals, and it 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 slave interface, which costs 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. Additionally, 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 state of the art SoC system bus – AMBA (Advanced Microcontroller Bus Architecture) [108] 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.
- Complexity of WISHBONE system bus architecture is lower, and the circuit area is relatively smaller.
- WISHBONE inter-connection is loyalty free. Additionally, many WISHBONE compatible free IPs have been developed and can be adopted to implement particular functions in SoC design.

Because of simplicity, flexibility, and loyalty free of WISHBONE, the proposed CABAC encoder is equipped with WISHBONE system bus interfaces to enable easy 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

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 Partitioning of WISHBONE System Bus Interfaces

Function of input packets receiving and coded results output can be implemented in one or two WISHBONE interfaces. For single interface design, circuit area is reduced as interface signals are shared by both input and output procedures; and it requires only single channel of system bus. However, in each clock cycle, only read or write operation can be performed at the bus interface, and in the case that unit BP needs to output packet and unit BN need to read input packet, one of the two operations needs to be paused, and throughput of CABAC encoding pipeline is degraded. Furthermore, system bus occupation time is long. The reason is that: for a single master interface design, the master cannot estimate 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 thesis 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 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.

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

To enable easy integration of WISHBONE classic and registered feedback IP cores, it is required [104] 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.

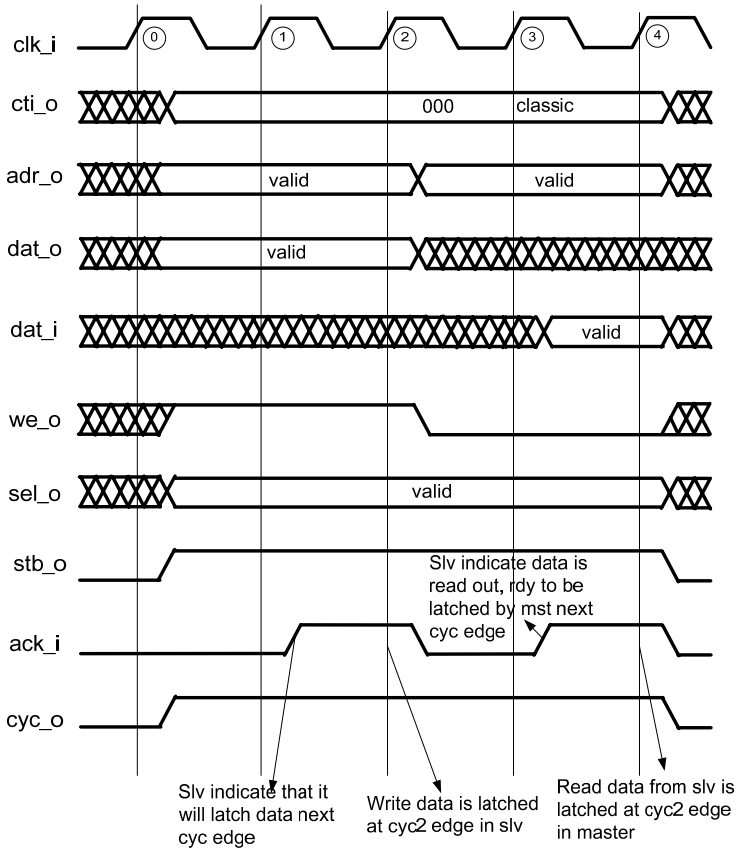


Figure 6-2: One classic cycle of a WISHBONE master interface with registered feedback of cycle termination.

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.

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 Figure 6-3.

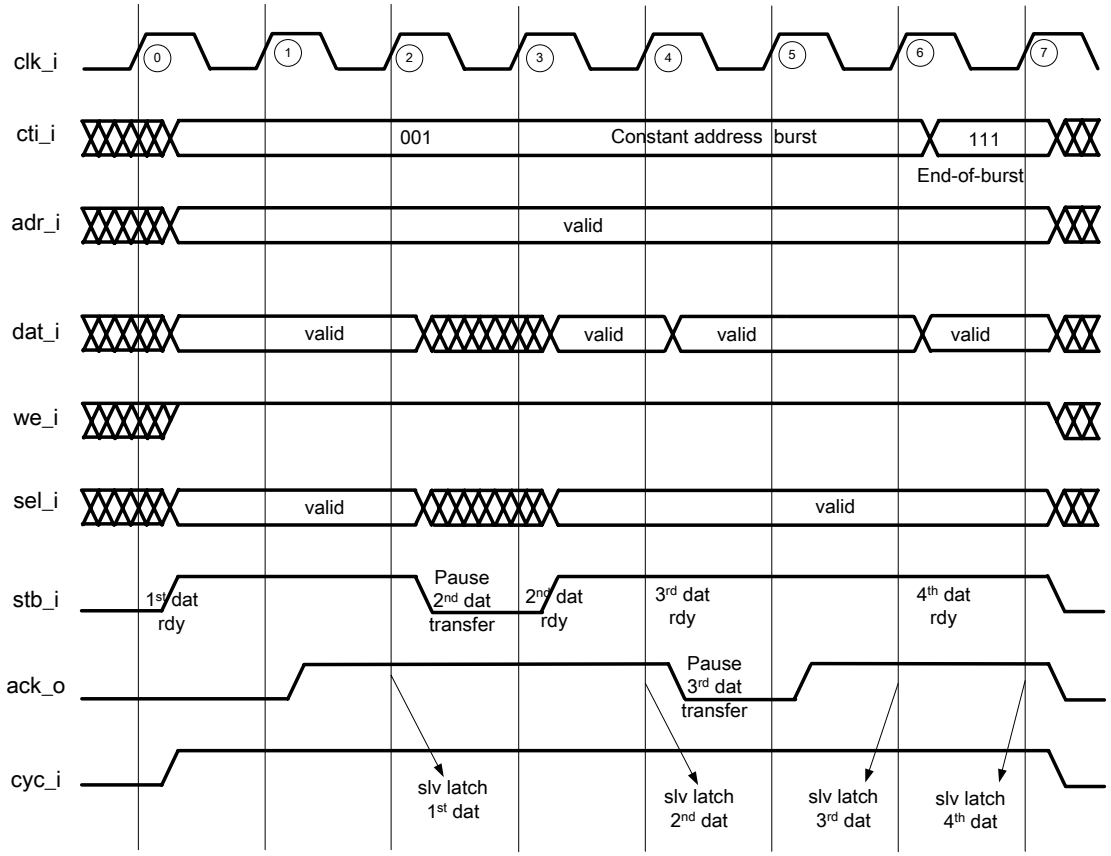


Figure 6-3: Illustration of constant address burst cycle of WISHBONE slave interface.

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 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 to 2 coded bytes can be output when *EOS_true* is 0, and 2 to 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.

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 to 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

Design of master interface with 32-bit data bus is shown in Figure 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.

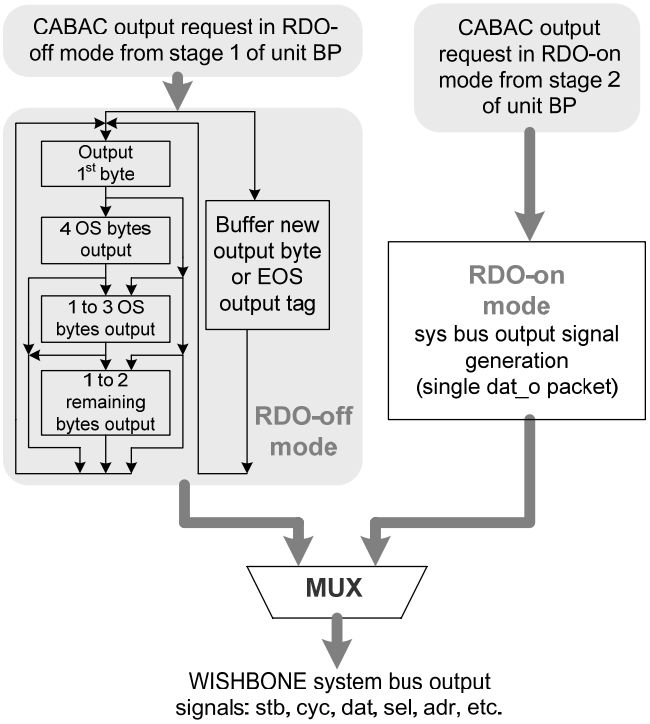


Figure 6-4: Data output control of WISHBONE master interface with 32-bit dat_o bus.

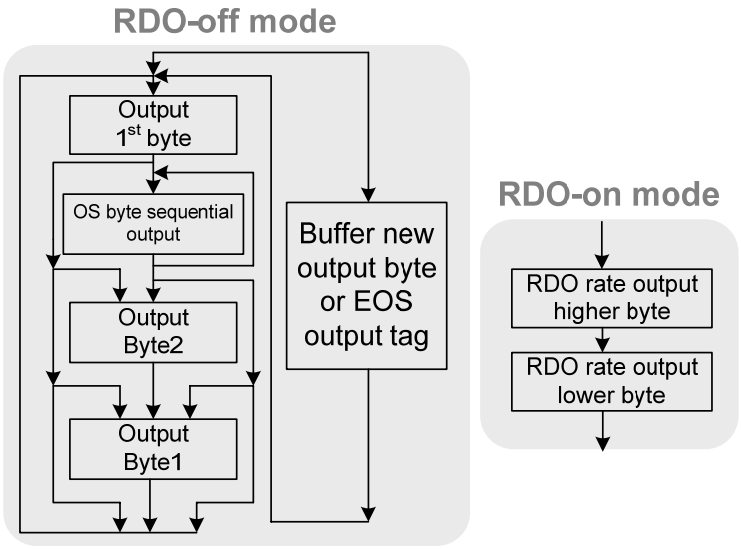


Figure 6-5: Data output control of WISHBONE master interface with 8-bit dat_o bus.

An alternative design of master interface is also implemented with dat_o bus width of 8 bits. As shown in Figure 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 to 7 packets in RDO-off mode. Control logic is simpler compared to that of master 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.

6.2.5 Consideration of Data Transfer Speed of System Bus

Long interconnection delay of system bus can limit maximal coding speed of CABAC encoder. Because of implementing data transfer of system bus interfaces as register feedback cycle, all inputs and outputs of CABAC are buffered with registers. Data transfer delay is only determined by transfer delay on the system bus. With proper insertion of buffers on the system bus similar to buffer insertion during clock tree synthesis, data transfer delay on the system bus will efficiently constrained with no

negative influence on CABAC encoding speed after integration of CABAC encoder IP in the SoC-based video coding system.

6.3 Design of System Bus Inter-connection (INTERCON)

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 [104]. 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 inter-connection. 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 Figure 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 Figure 6-6, the INTERCON consists of 4 sub-units (named M0, M1, M2, and M3) for the function of *master input selection & slave cyc generation* and 4 sub-units (named S0, S1, S2, and S3) for the function of *master connection arbitration & 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-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-S3, each of M0-M3 selects WISHBONE input signals for the master that is connected to the sub-unit.

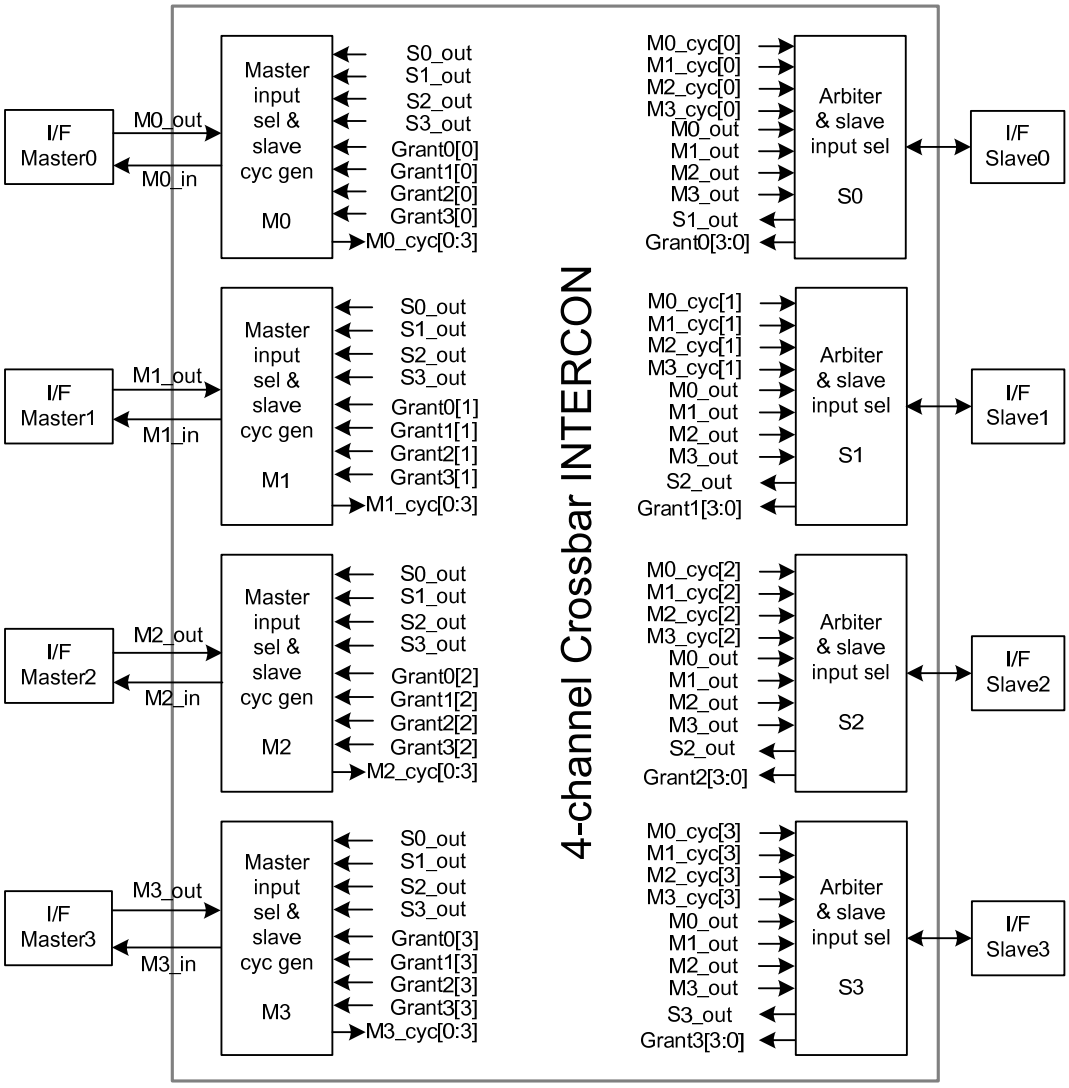


Figure 6-6: Top-level architecture of 4-channel crossbar INTERCON of WISHBONE system bus.

(1) Master Connection Arbitration & Slave Input Selection

The architecture of *master connection arbitration and slave input selection* sub-units (S0-S3) of INTERCON is shown in Figure 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.

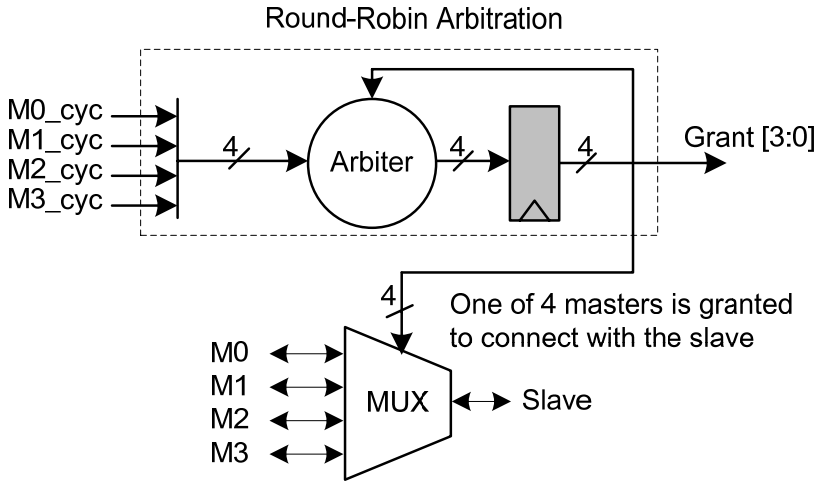


Figure 6-7: Round-robin arbitration of master that connects to the slave.

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 may need to wait until it receives a grant [109]. 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.

(2) Master Input Selection & Slave cyc Signal Generation

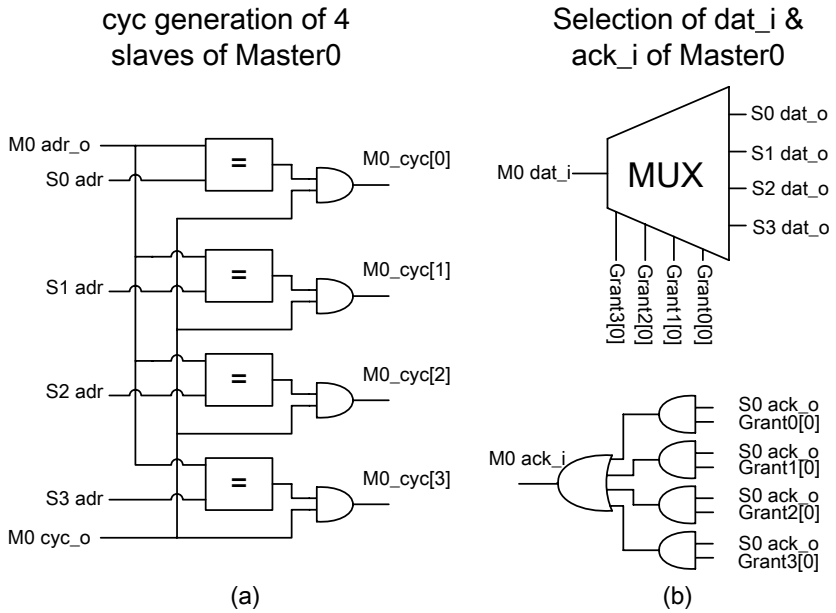


Figure 6-8: Architecture of M0 sub-unit: (a) Generation of cyc signals of 4 slaves that can connect to the master, and (b) selection of master input signal including dat_i and ack_i.

Architecture of master *input selection & slave cyc generation* sub-units (M0-M3) of INTERCON are illustrated in Figure 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 (Figure 6-8(a)), 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 Figure 6-8(b), 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

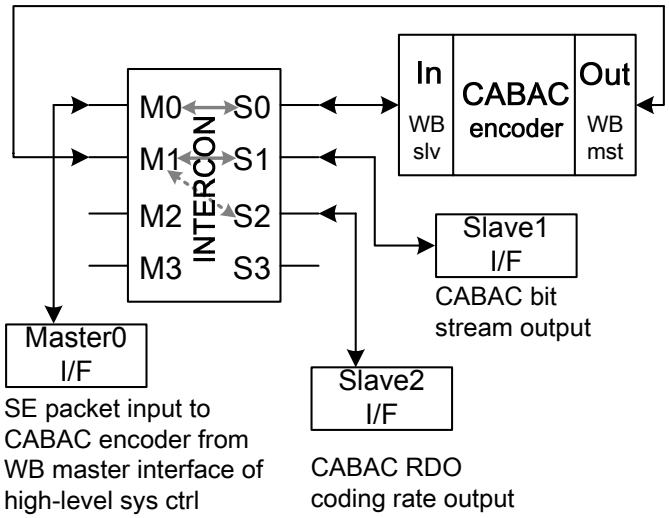


Figure 6-9: A compact inter-connection of CABAC encoder with other components of video encoder.

A compact SoC-based CABAC encoding system is shown in Figure 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 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 & 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 Design, Synthesis, and Performance Comparison

In this chapter, 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 & physical design. Then, contributions of other team members such as MBIST circuit insertion are briefly discussed for completeness. 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 & 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 previous 3 chapters are implemented and verified in 7 design steps, as illustrated in Figure 7-1. The step-by-step design & 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.

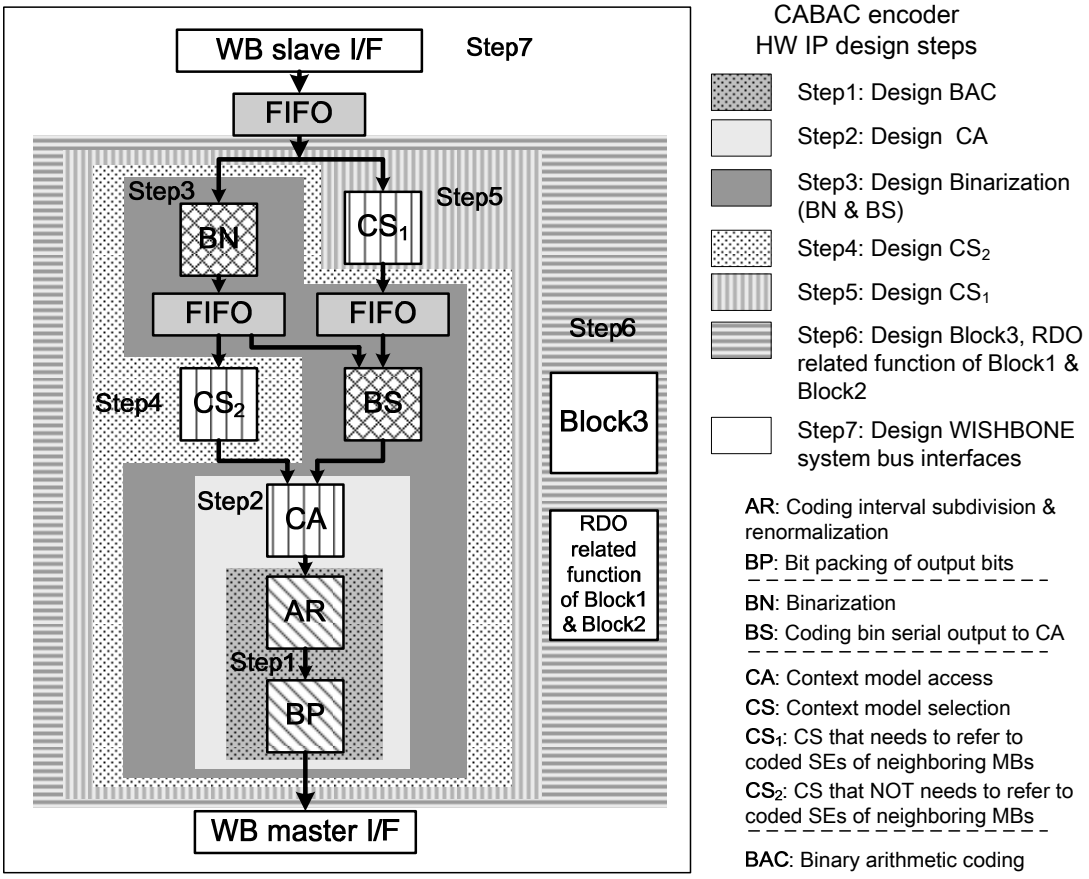


Figure 7-1: Design steps of CABAC encoder.

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 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. Functions 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 & coding interval and RDO coding rate output. In the last step (Step7), 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 encoder. After RTL-level function implementation in the 7 design steps, synthesis and physical design are carried out targeting at two types of foundry process technologies, and the results will be given in the next section.

7.1.2 Functional Verification of CABAC Encoder

The encoder is verified using two sets of video sequences: CIF (352×288) and HDTV 720p (1280×720), and each set of sequences contains 1 or more GOPs (Group of Picture), 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.

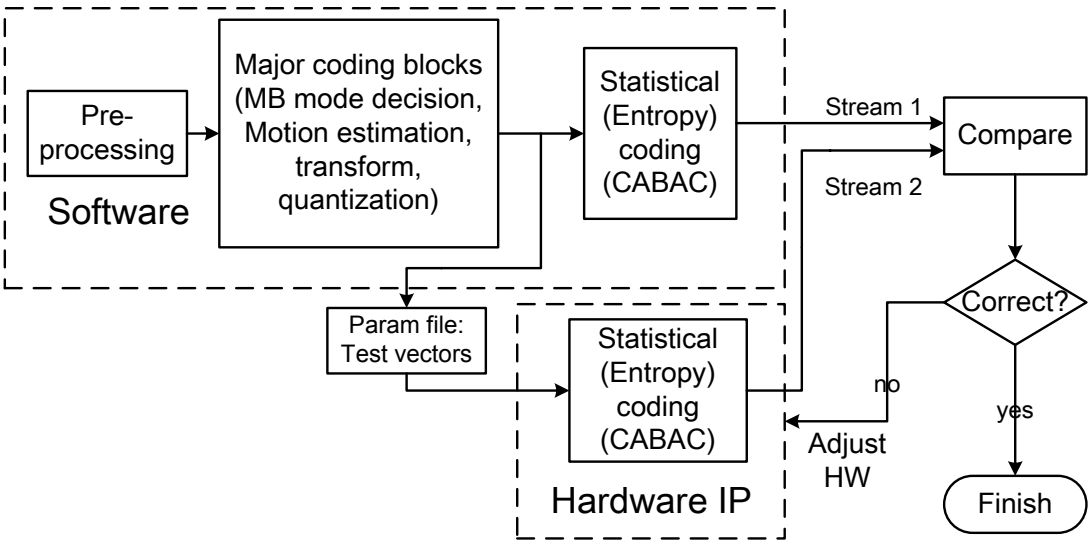


Figure 7-2: Verification of the HW IP block.

Verification of CABAC encoder design is illustrated in Figure 7-2. Two bit streams are generated and compared with each other. Stream 1 is generated from the reference SW [97] without any HW-assisted circuits using a video test sequence.. It can contain 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.

(1) Verification at Different Design Stages

The input to the CABAC SW encoder is redirected to a parameter file of test vectorswhich is then used as input of CABAC HW encoder, assuming the same system-level host processor. For each design step discussed previously, different sets of test vectors are utilized to verify encoder function. Detailed descriptions of required parameters of input test vectors at each design step are listed in Table 7-1.

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 & BP)	Bin packet containing bin, bin type, and context model
Step2	unit CA	Bin packet consisting of bin, bin type, CtxIdx
Step3	unit BN & BS	Packet of SE type & value, CtxIdx of regular bin
Step4	unit CS_2	Packet of SE type & 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 & value
Step6	Block3	Packet of SE type & value, Block3 ctrl parameters (ctx initialization, RDO ctx state backup & restoration), and RDO coding rate accumulation and output
Step7	Sys Bus I/F	dat_i of WISHBONE sys bus contains packet of step6

In order to accelerate verification of complex logic, excluding 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.

(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 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 consistence utilizing netlist and SDF (standard delay format) file that provides timing delay information of cells and connecting wires.

FPGA 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 thesis in addition to the ASIC design to verify design prototype for the entire CABAC encoder design with system bus interfaces integrated. Xilinx ISE [110] and Modelsim [111] are adopted to verify the encoder targeting at Virtex-II Pro FPGA at implementation steps of synthesis, mapping, and place & route. After logic simulation targeting at Virtex-4 library, bit stream of the design is generated and downloaded to the FPGA. ChipScope of Xilinx [112] is utilized to insert testing probes on the design before generation of bitstream, monitor probe values during hardware emulation, and transfer coded results to computer throughput USB cable. Design is successfully verified in two test cases. In case I: testing block and CABAC encoder are implemented in Single FPGA chip, and 100 MHz clock frequency is achieved. In case II, as shown in the following figure: two 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 & interconnection wires. As the design is fully verified, the encoder can be integrated directly into FPGA video coding system.

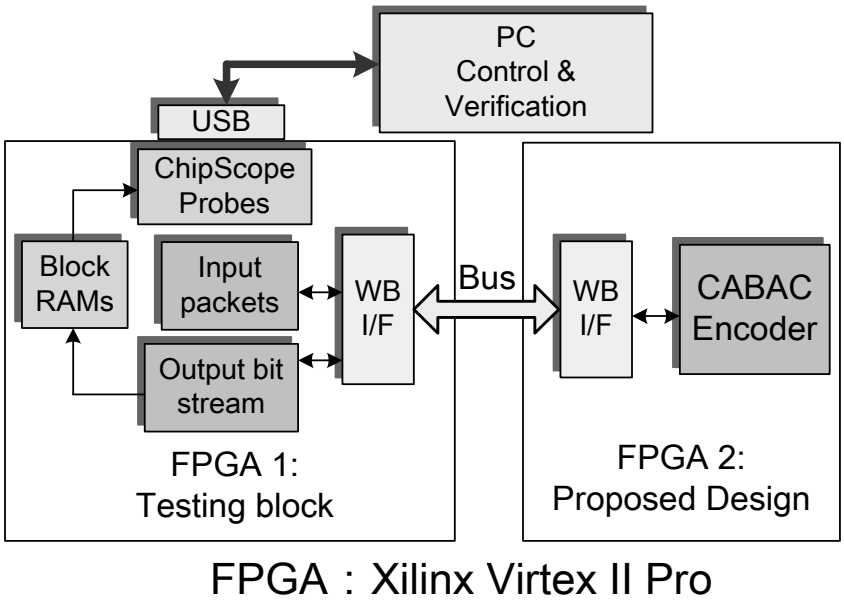


Figure 7-3: FPGA implementation and verification platform

7.2 Results of Synthesis and Physical Design

The scope of this research includes logic synthesis of the SoC-based CABAC using Design Compiler [113], and not the physical layout of the proposed design. Thanks to Xi Jiang, the proposed CABAC encoder was laid out using TSMC 0.13 μm process to the GDS-II stage targeting at TSMC 0.13 μm process using Astro [114]. Because of significant differences of implemented functions of reported designs and proposed encoder, functional completeness of CABAC encoder designs are analyzed first, before the result discussion of synthesis and physical implementation.

Most of reported designs only implement functions of context model access (CA) and BAC or only implement BAC, which are only part of functions of Block 2 of the proposed CABAC encoder, and it means that 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 per cycle and fast RDO context state backup & restoration operations in Block 3. Binarization and context model selection of CABAC were not implemented in [86, 87, 93, 105], and it was partially implemented in [95].

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 & 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 [93] that focuses on HW assisted RDO only supports context state backup & 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 (bin/cycle), synthesis/layout maximum operating frequencies (cycle/s), and areas (number of 2-input NAND gates or mm^2) of the proposed design and other reported designs targeting at CMOS process technologies or FPGA are compared in Table 7-2.

Table 7-2: Encoding pipeline throughput, max frequency, area of CABAC encoders

Design	**Process technology**	**Bin/ cycle**	**Clock Freq MHz**	**Circuit area of the implemented functions**		
Li06 [86]	0.35 μm syn. ROHM	0.59	150	4.57K gates (only part of Block 2) 4.1Kb RAM		
Kuo06 [87]	0.18 μm syn. TSMC	1	200	0.31mm^2 (only part of Block 2)		
Chen07 [89]	0.15 μm syn.	0.56	333	13.3K excluding mem (CtxIdxInc calc. at host)		
Liu07 [90]	0.13 μm syn. TSMC	0.67	200	34.3K excluding mem		
Nunez06 [93]	XilinxV4 FPGA	1	130	1158 slices (encoder only for part of block 2: 856 slices)		
Osorio06 [95]	0.35 μm syn. AMS	1.9 ~ 2.3	186	19.4K excluding mem (Block 2, partial of binarization)		
Shojania05 [105]	0.18 μm syn. TSMC	0.33	263	0.423 mm^2 (only part of Block 2)		
This design	0.13μm TSMC *post-layout*	1	328 (*worst-case*)	1.41 mm^2	Including: WB I/F, all RAMs, ROMs	
				Total	unit CS_1	Block 3
	0.13 μm syn. TSMC		578	44.6K	15.2K	8.9K
	0.35 μm syn. AMS		186	31.2K	11.1K	5.7K

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, as aforementioned. Reported designs that implement similar encoder functions are compared. The proposed functional Block 3 – unique to all reported designs – takes up 8.9K gates (20.0% of circuit area) in 0.13μm process; while unit CS_1 of this design occupies 15.2K gates (34.1% of circuit area) in the same technology, because large number of registers is allocated to store 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 & BAC, and function of binarization and context model selection of CS_2 utilize 18.6%. Only design in [90] supports similar function of unit CS_1 with 15.9K gates of the block. However, in [90], RDO is not supported in the block, and SE access delay of neighboring MB is significant. In terms of throughput, those of [86, 89, 90, 105] are lower than 1 bin/cycle, while those of [87, 93], and the proposed design are 1 bin/cycle, and [95] is at round 2 bin/cycle. If the *maximum number of bins processed per second* is a determining factor, the proposed design offers higher processing speed (bin/s) compared to[86, 87, 89, 90, 105]. Design [95] reportedly offers the highest processing speed. Performance of the proposed encoder and [95] 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 & LVS checking. The chip layout is shown in Figure 7-4. 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 (1.08V, 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.

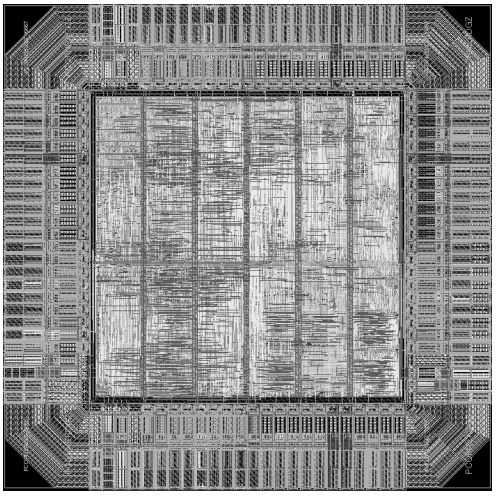


Figure 7-4: Chip Layout of the CABAC Encoder.

7.3 Power Reduction Strategies & 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 Chapter 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 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 [115]. Dynamic power of the design is significantly reduced with power reduction techniques applied.

Available power consumptions of reported designs ([87], [105]) and proposed design (RDO-off mode) are precisely compared using same TSMC 0.18μm CMOS process at same clock frequency of 200 MHz, as listed in Table 7-3. For the same function block of Block 2 and Normal RAM access, power consumptions of [87], [105], and this design are 20.7 mW, 36.5 mW, and 19.6 mW, respectively. This design achieves power reduction of **6%** and **46%** compared to [87] and [105], indicating that power is more efficiently constrained compared to Kuo's low power cache-based design.

Table 7-3: Gate-level power consumption (mW) of reported designs and proposed design

Design	**Total power**	**Power of Block 2 and Normal context RAM access**	
Kuo06 [87] (partial of Block 2 func & RAM acc)	N/A		**20.7** (200MHz, 0.18μm)
Shojania05 [105] (partial of Block 2 func & RAM acc)	N/A	48.0 (263MHz, 0.18μm)	**36.5** (263MHz, 0.18μm)
This design at 200Mbps, 0.18μm TSMC RDO-off mode	48.9	25.7 (w RDO)	**19.6** (w/o RDO)

For the power consumption on the host processor, low power embedded processors typically consumes 0.2 mw to 2 mw per MIPS [116], such as the ARM 9-11 series and the PowerPC. In the instruction-level analysis of H.264/AVC reference SW encoder [97] in QP 12 to 28, it is found that 58.8% of CABAC computation still need to be calculated by the host processor in [87] and [105], 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 in TSMC 0.13 μm CMOS process is evaluated in several video coding configurations, as shown in Table 7-4. 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.

Table 7-4: 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 7-5: 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%

The average distribution of power consumption of the proposed CABAC encoder in different function blocks are listed in Table 7-5 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 7-5 illustrates that percentage of power consumption of Block 3 and Memory block increases in RDO-on mode. It is because in RDO-on mode, Block 3 and context RAMs are more frequently accessed for context state backup & 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 & 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.

7.4 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 [117] 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.

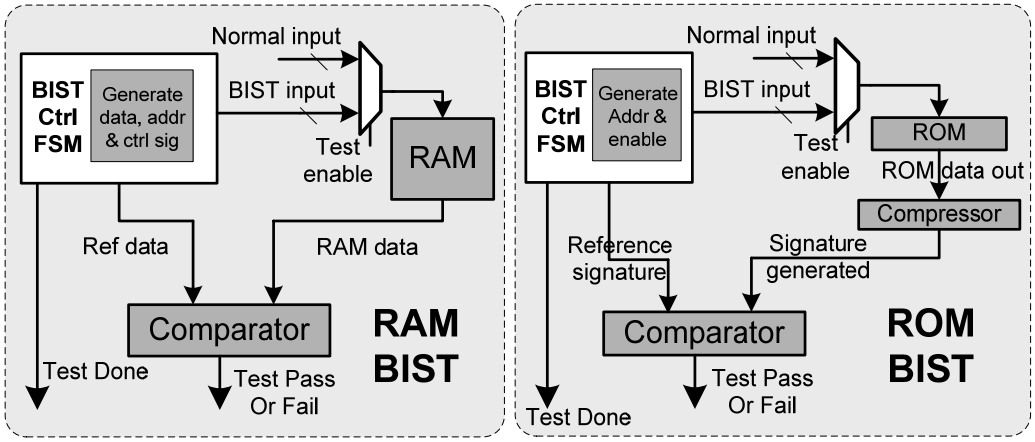


Figure 7-5: BIST testing circuits of memory block, including RAM BIST and ROM BIST.

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 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 Figure 7-5. 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 Figure 7-5, 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 & Temp RAM, 1 for Best RAM and P8x8 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.

7.5 Performance Comparison

In the following subsections, 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.

7.5.1 CABAC Encoding Speed Performance of the Encoder

(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 (7-1).

$$t_{CABAC\_top} << t_{ref\_SW} \tag{7-1}$$

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 [118] 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 synthesis and physical design tool. Therefore, the expression for number of cycles corresponding to (7-1) can be derived as:

$$c_{CABAC\_top} << c_{ref\_SW} \tag{7-2}$$

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:

$$c_{CABAC\_top} = c_{HW\_IP} + c_{SW\_nonIP} + c_{transfer\_latency} \tag{7-3}$$

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 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 (7-3), 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 7-6, 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.

Table 7-6: 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

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.

(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. Additionally, 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 thesis, the problem of residual SE generation is released by (a) input run-level pairs instead of all coefficients of residual block to reduce 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 to 32. As shown in Table 7-7, 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 [90] and [95] 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 section 7.5.3.

Table 7-7: Average throughput of the proposed CABAC encoder in video coding tests

QP	12	16	20	24	28	32
Total encoding cycles	1874123	1103743	566390	314376	190255	117383
bin number	1855991	1096048	562211	311756	188186	115195
Avg throughput	0.990	0.993	0.993	0.992	0.990	0.981

(3) Worst-Case Analysis of Proposed CABAC Encoder

For the worst-case analysis, the H.264/AVC encoder is tested in HDTV 720p 60fps 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 (1.08V, 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.

7.5.2 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 & write access; and (2) efficiency of context state backup & restoration operation. The performance is compared with [93], which is the only reported design that fully supports different RDO context state backup & restoration operations. The influence of context RAM reallocation is also evaluated.

(1) Comparison *of Context RAM Read & Write Access Efficiency*

CABAC encoder is tested using 4 CIF video sequences: Foreman, Mother & 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 & write access frequency ratio of this design over [93] 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 [93] (*access ratio*) in the QP range of 12 to 32 is shown in Table 7-8. With techniques of both context line access scheme and context RAM reallocation applied, the context RAM access frequency of this design is 26.1% to 33.2% of that of [93] in the 4-sequence tests, with an average of 28.4%. The RAM access frequency of this design is significantly lower than [93].

Table 7-8: Average context RAM access frequency ratio (This design over [93] 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%

Applying the context line access scheme alone, the average access ratio of this design over [93] is 33.9%. After context RAM reallocation, the average RAM access frequency of this design further reduces 16.1%, as shown in Table 7-9, indicating that the reallocated context RAM efficiently reduces context RAM access frequency.

Table 7-9: 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%

Figure 7-6 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.

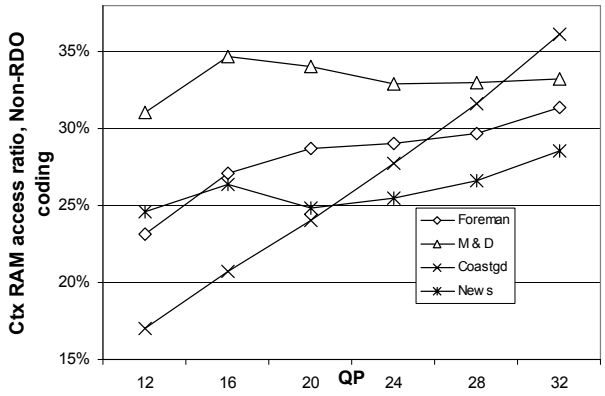


Figure 7-6: Context RAM access frequency ratio of this design over [93], during RDO-off coding in the QP range of 12 to 32 of 4 typical video sequences.

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

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.

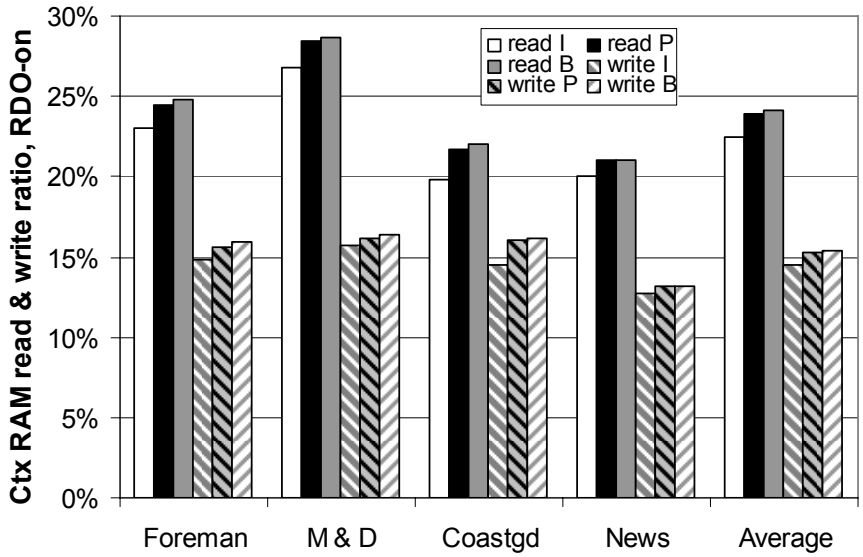


Figure 7-7: Context RAM read and write frequency access ratio of this design over [93], during RDO-on coding. The average access ratios of I, P, and B frames of 4 video sequences in QP range of 12 to 32 are shown.

Figure 7-7 shows the average RAM read frequency ratio (read ratio) and RAM write frequency ratio (write ratio) of this design over [93] in the QP of 12 to 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.

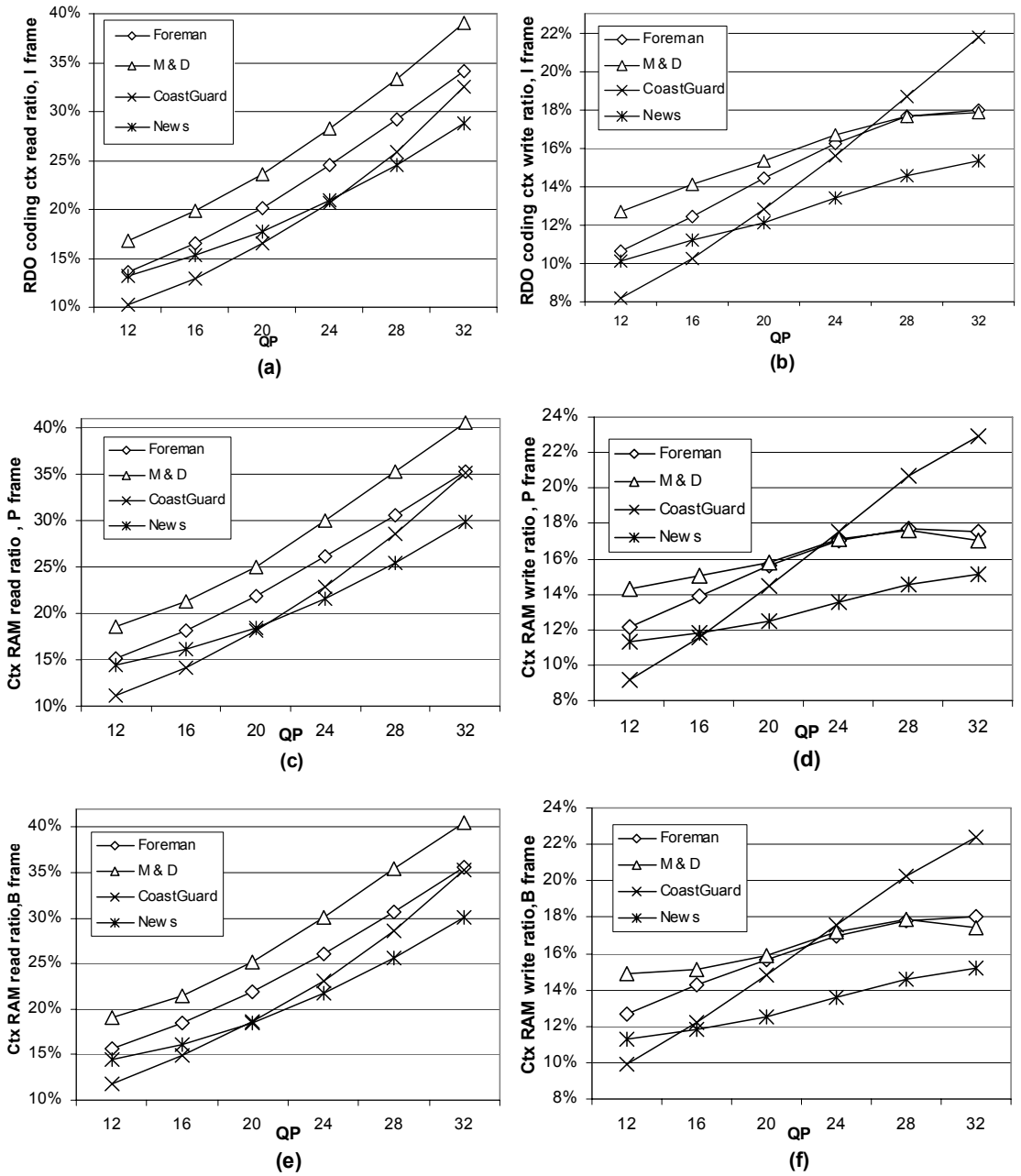


Figure 7-8: Context RAM access frequency ratio of this design over [93] during RDO coding in the QP range of 12 to 32 of 4 video coding sequences. Read ratio of I, P, and B frames are illustrated in (a), (c), and (e) respectively; Write ratio of I, P, and B frames are illustrated in (b), (d), and (f).

Figure 7-8 shows the read ratio and write ratio of this design over [93] 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 Intra4x4 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 [93] and other designs with single context model access scheme. The widely used industry SRAM model CACTI 5.3 of HP lab (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 53-wordx56-bit of this design and 400-

wordx8-bit of reference. According to SRAM access frequency ratio of this design and [93], 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.

(2) Efficiency of Context *State Backup & Restoration (B&R) Operation*

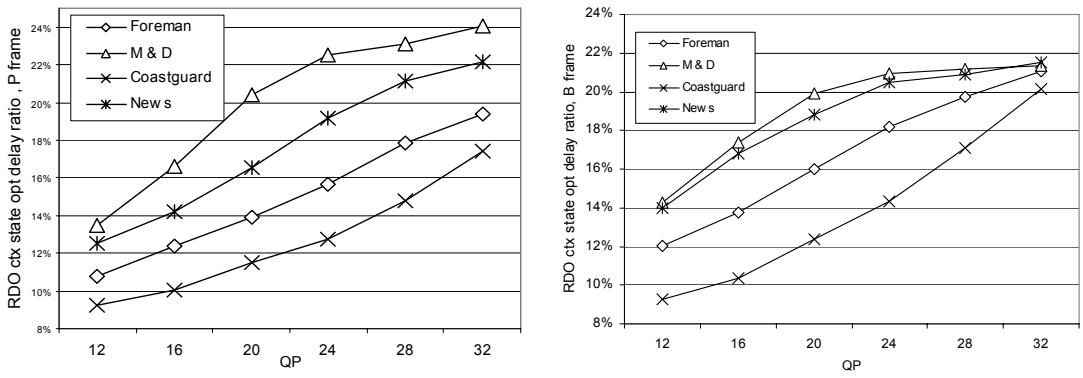
Compared to [93], the operations of context state restoration are removed during non-P8×8 RDO coding, because of allocation of Temp RAM to store updated context lines during coding of each RDO mode, as discussed in Chapter 5. During RDO-on coding of P8×8 sub-MB mode decision, 4 partition modes 8×8, 8×4, 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 [93], 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 & 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 [93]. 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 [93].

Average P8×8 context state B&R operation delay ratio *ctx_opt_delay_ratio* of the proposed encoder to [93] is evaluated by (7-4). *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 [93] is evaluated as the ratio of the context lines accessed in proposed design (*ctx_line_num*) to the context models accessed in [93] (*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 (7-4).

$$ctx\_opt\_delay\_ratio = \frac{ctx\_opt\_frm}{20 \times MB\_num\_frm} \times \frac{ctx\_line\_num}{ctx\_model\_num} \tag{7-4}$$



(a) (b)

Figure 7-9: Context state backup & restoration operation delay ratio of this design to [93] in P8×8 RDO coding for QP 12 to 32 of 4 video coding sequences. Ratio of P frame coding in (a) and ratio of B frame in (b).

Figure 7-9 illustrates the timing delay ratio of this design to [93] 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 (7-4) 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 & 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 [93] for P and B frame coding of in 5 CIF sequences coding tests, as shown in Table 7-10.

Table 7-10: Average context state backup and restore operation delay ratio of the proposed design to [93]

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%

(3) Context RAM Occupation Reduction

In [93], 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. Additionally, 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 [93], only 7.37 Kbits of context RAMs are allocated in this design, which is 16.0 % of [93]. 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.

7.5.3 Performance Comparison with the State-of-the-Art Design

(1) Function Completeness

Table 7-11: Functional comparisons of [95] and the proposed design

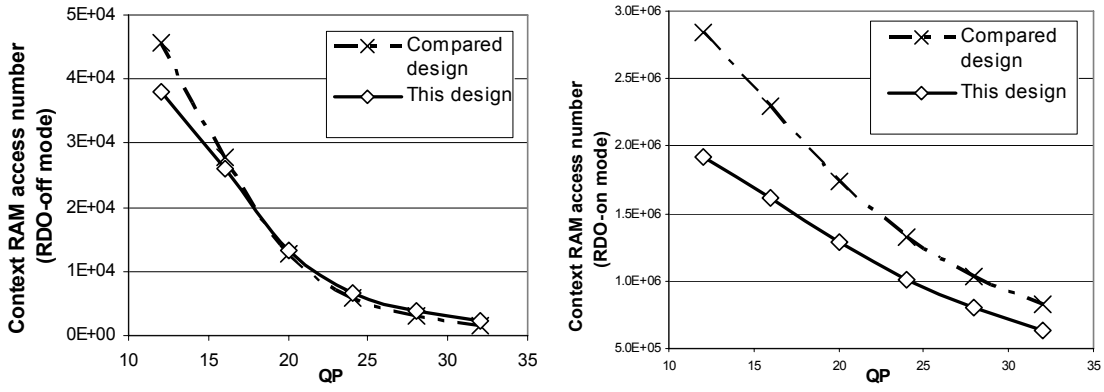
Function difference	[95]	This design
Binarization for non-residual SE	No	Yes
CtxIdxInc calculation (context model selection) for regular bins	Only support for SCF, LSCF, level	Full support for all types of SE
Additional bin pairing process for non-residual SE in the host processor	Yes	No
Input of residual 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

Design [95] 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 & 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 [95] and the proposed design are summarized in Table 7-11.

For non-residual SE and CBF processing of [95], 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 Chapter 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 [95], 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 [95], especially in high QP coding, in which the ratio of non-zero residual coefficients are low. In [95], 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 [95], 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 [95] 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 [95], 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.

(2) Context RAM Access Efficiency



(a) (b)

Figure 7-10: Average context RAM access number per frame of residual SEs in [95] (compared design) and this design in CIF frame coding for QP 12 to 32. The access numbers of RDO-off coding and RDO-on coding are shown in (a) and (b), respectively.

Table 7-12: Context access performance (number of RAM access) of the proposed encoder compared to [95] in residual SE coding

Test cond.	RDO	Design	QP 12	QP 16	QP 20	QP 24	QP 28	Avg. %↓
Avg. of 1frame of one GOP	Off	This	38026	26015	13283	6757	3725	-3.3%
		[95]	45542	27788	12796	5932	3059	
		%↓	16.5%	6.4%	-3.8%	-13.9%	-21.7%	
	On	This	1919744	1620159	1287209	1015775	809390	26.7%
		[95]	2840789	2299346	1743681	1323572	1039464	
		%↓	32.4%	29.5%	26.2%	23.3%	22.1%	
First intra frame	Off	This	48754	40412	31237	21934	15322	7.1%
		[95]	62428	48178	33743	21697	14086	
		%↓	21.9%	16.1%	7.4%	-1.1%	-8.8%	
	On	This	1754173	1549247	1305428	1067836	873608	28.7%
		[95]	2701221	2293064	1828704	1419576	1126159	
		%↓	35.1%	32.4%	28.6%	24.8%	22.4%	

This: The proposed encoder of this thesis.
%↓: Percentage of encoding cycles reduction, proposed design compared to [95].

Design [95] 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 [95] and this paper can be compared. In

Figure 7-10, 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 (Figure 7-10 (a)) and RDO-on mode (Figure 7-10 (b)), and the context RAM access numbers of two designs in QP 12 to 28 are listed in Table 7-12.

In RDO-off coding, the averaged RAM access efficiency of [95] 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 [95]. Because this CABAC design is targeting at high quality and high bit-rate coding, it is beneficial to have lower RAM access frequency in this QP range. Table 7-12 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 Figure 7-10 (b) and Table 7-12. It is because in the RDO-on mode of [95], 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 [95] 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 [95] 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 & restoration in P8×8 RDO coding and related context RAM size are not comparable because the operation is not supported in [95].

(3) SoC-based CABAC Encoder IP

Compared to [95] 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 rate, and compression ratio. In video coding test of the proposed encoder, to support RDO-off coding at 16Mbps 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 97.3M packet/s, and output RDO rate is 11.3M packet/s.

Chapter 8 Conclusions

In this thesis, the design and implementation of a high-performance CABAC encoder targeting at the Main profile of H.264/AVC video coding standard has been carried out with success. The main purpose of the research is to accelerate serial CABAC encoding procedure in different video coding configurations and remove the bottleneck of H.264/AVC encoding at statistical coding stage by the approach of 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) in different RDO modes, efficient context model access, and low power consumption.

8.1.1 Summary of Design Advantages

Generally speaking, this work enhances the performance of both CABAC encoder and the H.264 video coding system. It achieves global performance optimization compared to local optimization of particular HW function blocks such as CA, BAC, and residue SE coding in most of reference designs. Computation on the host processor and data transfer of system bus are minimized, and more importantly, this work is the first reported design that successfully solves design difficulty of CABAC coding in RDO mode and provides best video compression efficiency of H.264/AVC standard. Power consumption of this high performance encoder is also lowest for both HW and system power in the reported designs. Flexibility of encoder design is utilized by selection of proper encoding functions to achieve targeting performance. This is different from decoder design, and it

is ignored by most of other designs. It is the only design that provides a complete SoC-based IP solution for CABAC encoder that can support different types of video coding configurations such as RDO-off coding, fast RDO and full RDO coding, etc. The application range of the design is wider, from real time coding to high quality compression. This IP is reusable and suitable for future video coding system. Considering encoding speed, it is the only CABAC encoder that achieves real time CIF coding in full RDO mode and HDTV coding in RDO-off and fast RDO mode. The major design advantages of the proposed CABAC encoder IP of the thesis 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 function unit design - context model selection of context modeling is efficiently designed, which achieves single cycle context model selection. 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; (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.

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 & 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 & 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 × clock frequency) of the CABAC encoder. Full pipelined architecture of bin encoding is designed that can process 1 bin packet per cycle. Functional partitioning of encoding pipeline stages of bin packet generation, context model access, coding interval subdivision & renormalization, and bit packing efficiently reduce the data dependency of the sequential coding steps. 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 [95] and [90]. Additionally, circuit critical path is significantly reduced and higher encoder clock frequency is achieved, compared to most reported designs.

4. Efficient Context Model Access

An efficient context model access scheme of CABAC encoder is proposed in this thesis, including techniques of context line access and buffering, context memory reallocation, and pipelined context model B&R operation in P8×8 RDO coding. Context memory size is reduced to 16.0% of [93]. Context RAM read and write access frequency in both RDO-off and RDO-on coding modes are significantly lower than [93]. Context state backup & restoration operation delay of P8×8 RDO coding mode is 15.5% and 16.6% of [93] in P and B frame coding tests, while the operation of non-P8×8 coding is removed. With the reduction of memory access frequency, power consumption of context RAM blocks is also lower, especially in RDO coding with power reduction of 31% (read) and 56% (write). Compared to the cache-based context model access of [95], context model access frequency of the proposed design is significantly lower in RDO-on mode, and cache miss data fetch delay 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 HW power is lower than references (46% and 6% reduction compared to two references including a cache-based low power design) for the same function blocks with same 0.18μm process technology and clock frequency. Total power consumption of CABAC encoding on the host processor and HW encoder is even lower. 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 is 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 thesis utilizing different design strategies to achieve complete function features, high & constant coding throughput, and improved reusability and portability. This design is verified, synthesized, and laid out at the GDS-II stage with post-layout speed suitable for high quality real time video coding.

Several design strategies utilized for the proposed CABAC encoder of this thesis are also suitable for similar R&D projects of serial coding and highly data dependent system. The strategies include:

- Widely used pipeline architectures in the operations of bin encoding, context state backup & 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 $Range_{LPS}$ 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 at the Main profile of H.264/AVC standard, it can be easily scaled to the High profiles of the standard, because of very similar implementation schemes and design architectures of Main profile and High 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.

8.1.2 Future Research Directions

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) The direction discussed in VCEG for the next generation of video coding

standard: 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.

The current design follows SoC-based HW/SW design flow with SW/HW interface defined and HW IP implemented through RTL level, gate level, FPGA implementation, and physical design stage. FPGA-based design is flexible with lower risk compared to ASIC tape out flow. However, coding speed is limited by the logic and memory volume of FPGA chip and longer interconnection delay of FPGA chip. In the future design, it is possible to use large volume FPGA chip fabricated with new process technology that can provide enough HW resources and speed for real time CABAC encoding in high complexity HDTV video coding. It is beneficial to avoid using platform-specific FPGA IP cores, so that same RTL design can be easily used in different FPGA chips. Because it is more suitable to implement high level video coding control in SW, it is necessary to select proper FPGA chip with high performance HW processor core and SoC SDK tools to enable system integration of processor and HW IP through on-chip system bus.

As discussed above in research direction (3), parallel CABAC encoding can be explored in multi-core or many-core processor platform or in multiple CABAC HW encoding cores. However, CABAC encoding and decoding algorithms need to be revised in order to break data dependency of coding states and enable processing of multiple entropy slices inside each slice. It is possible to accelerate CABAC codec by parallel coding scheme if the technique is accepted by the next generation of video coding standard. Compared to the scheme with multiple processor cores, scheme with multiple HW cores

is more efficient in coding speed enhancement. Cautious update of CABAC algorithm is necessary to minimize compression efficiency loss of parallel processing by exploring strategies using available context information.

Bibliography

[1] "Video Codec for Audiovisual Services at px64 kbit/s," ITU-T, ITU-T Recommendation H.261, Version 1, 1990.

[2] "Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at Up to About 1.5 Mbit/s," ISO/IEC JTC 1, ISO/IEC International Standard 11172 (MPEG-1), 1993.

[3] "Information Technology - Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video," ITU-T and ISO/IEC JTC 1, ITU-T Recommendation H.262 and ISO/IEC International Standard 13818-2 (MPEG-2 video), 1994.

[4] "Video Coding for Low Bit Rate Communication," ITU-T, ITU-T Recommendation H.263 version 1, 1995.

[5] "Information Technology - Coding of Audio-Visual Objects—Part 2: Visual," ISO/IEC JTC1, ISO/IEC International Standard 14496-2 (MPEG-4 Visual Version 1), 1999.

[6] "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.

[7] "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.

[8] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.13, no.7, pp. 560-576, 2003.

[9] G. Bjøntegaard and K. Lillevold, "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, 2002.

[10] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.13, no.7, pp. 620-636, 2003.

[11] G.J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine*, vol.15, no.6, pp. 74-90, 1998.

[12] L. Yu, J. Li, and Y. Shen, "Fast Frame/Field Coding for H.264/AVC," *in Proceedings of International Conference on Digital Telecommunications*, pp.18-18, 2006.

[13] G.J. Sullivan and R.L. Baker, "Rate-distortion optimized motion compensation for video compression using fixed or variable size blocks," *in Proceedings of Global Telecommunications Conference*, pp.85-90 vol.1, 1991.

[14] D. Marpe and T. Wiegand, "A highly efficient multiplication-free binary arithmetic coder and its application in video coding," *in Proceedings of International Conference on Image Processing*, pp.II-263-266 vol.3, 2003.

[15] Z. Wei, K.L. Tang, and K.N. Ngan, "Implementation of H.264 on Mobile Device," *IEEE Transactions on Consumer Electronics*, vol.53, no.3, pp. 1109-1116, 2007.

[16] S. Kant, U. Mithun, and P.S.S.B.K. Gupta, "Real time H.264 video encoder implementation on a programmable DSP processor for videophone applications," *in Proceedings of International Conference on Consumer Electronics*, pp.93-94, 2006.

[17] H.-C. Lin, Y.-J. Wang, K.-T. Cheng, S.-Y. Yeh, W.-N. Chen, C.-Y. Tsai, T.-S. Chang, and H.-M. Hang, "Algorithms and DSP implementation of H.264/AVC," *in Proceedings of Asia and South Pacific Conference on Design Automation*, 2006.

[18] J. Lahti, J.K. Juntunen, O. Lehtoranta, and T.D. Hamalainen, "Algorithmic optimization of H.264/AVC encoder," *in Proceedings of IEEE International Symposium on Circuits and Systems*, pp.3463-3466 Vol. 4, 2005.

[19] H. Baik, K.-H. Sihn, Y.-i. Kim, S. Bae, N. Han, and H.J. Song, "Analysis and Parallelization of H.264 decoder on Cell Broadband Engine Architecture," *in Proceedings of IEEE International Symposium on Signal Processing and Information Technology*, pp.791-795, 2007.

[20] T.-Y. Huang, G.-A. Jian, J.-C. Chu, C.-L. Su, and J.-I. Guo, "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*, pp.2189-2192, 2008.

[21] F. Pan, L.S. Rahardja, K.P. Lim, L.D. Wu, W.S. Wu, C. Zhu, W. Ye, and Z. Liang, "Fast intra mode decision algorithm for H.264-AVC video coding," *in Proceedings of International Conference on Image Processing*, pp.781-784 Vol.2, 2004.

[22] H. Kim and Y. Altunhasak, "Low-complexity macroblock mode selection for H.264-AVC encoders," *in Proceedings of International Conference on Image Processing*, pp.765-768 Vol.2, 2004.

[23] M. Nieto, L. Salgado, and J. Cabrera, "Fast Mode Decision on H.264/AVC Main Profile Encoding Based on PSNR Predictions," *in Proceedings of IEEE International Conference on Image Processing*, pp.49-52, 2006.

[24] S.-N. Ba, Y. Altunbasak, and H. Ates, "Low Complexity Inter-Mode Selection for H.264," *in Proceedings of IEEE International Conference on Image Processing*, pp.1349-1352, 2006.

[25] Y. Li, Y. Qu, and Y. He, "Memory Cache Based Motion Compensation Architecture for HDTV H.264/AVC Decoder," *in Proceedings of IEEE International Symposium on Circuits and Systems*, pp.2906-2909, 2007.

[26] H. Schwarz and T. Wiegand, "R-D Optimized Multi-Layer Encoder Control for SVC," *in Proceedings of IEEE International Conference on Image Processing*, pp.II - 281-284, 2007.

[27] S. Ma, W. Gao, and Y. Lu, "Rate-distortion analysis for H.264/AVC video coding and its application to rate control," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.15, no.12, pp. 1533-1544, 2005.

[28] T.-C. Chen, S.-Y. Chien, Y.-W. Huang, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, and L.-G. Chen, "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.16, no.6, pp. 673-688, 2006.

[29] Y.-W. Huang, T.-C. Chen, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, C.-S. Chen, C.-F. Shen, S.-Y. Ma, T.-C. Wang, B.-Y. Hsieh, H.-C. Fang, and L.-G. Chen, "A 1.3TOPS H.264/AVC single-chip encoder for HDTV applications," *in Proceedings of IEEE International Solid-State Circuits Conference*, pp.128-588 Vol. 1, 2005.

[30] Z. Liu, S. Yang, S. Ming, L. Shen, L. Lingfeng, S. Ishiwata, M. Nakagawa, S. Goto, and T. Ikenaga, "A 1.41W H.264/AVC Real-Time Encoder SOC for HDTV1080P," *in Proceedings of IEEE Symposium on VLSI Circuits*, pp.12-13, 2007.

[31] K. Inata, M. Sasamoto, T. Nonaka, and H. Komi, "System Architecture of H.264/AVC Codec LSI for Digital HD Camcorder," *in Proceedings of International Conference on Consumer Electronics*, pp.1-2, 2008.

[32] L. Agostini, R. Porto, J. Guntzel, I. Saraiva Silva, and S. Bampi, "High throughput multitransform and multiparallelism IP for H.264/AVC video compression standard," *in Proceedings of IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006*, pp.4 pp., 2006.

[33] T.-C. Chen, C.-J. Lian, and L.-G. Chen, "Hardware architecture design of an H.264/AVC video codec," *in Proceedings of Asia and South Pacific Conference on Design Automation*, 2006.

[34] S. Lee, S. Park, J. Han, N. Eum, and P. Jongwon, "A 40MHZ dedicated hardware H.264/AVC video encoder with the reducing memory access scheme," *in Proceedings of IEEE International Symposium on Consumer Electronics*, pp.1-4, 2008.

[35] S.C. Chang, C.-C. Cheng, and L.-G. Chen, "System Architecture Design Methodology for H.264/AVC Encoder," *in Proceedings of IEEE International Symposium on Consumer Electronics*, pp.1-5, 2007.

[36] Y.-K. Lin, L. De-Wei, L. Chia-Chun, K. Tzu-Yun, W. Sian-Jin, T. Wei-Cheng, C. Wei-Cheng, and C. Tian-Sheuan, "A 242mW, 10mm2 1080p H.264/AVC high profile encoder chip," *in Proceedings of 45th ACM/IEEE Design Automation Conference*, pp.78-83, 2008.

[37] Y.-K. Lin, L. De-Wei, L. Chia-Chun, K. Tzu-Yun, W. Sian-Jin, T. Wei-Cheng, C. Wei-Cheng, and C. Tian-Sheuan, "A 242mW 10mm2 1080p H.264/AVC High-Profile Encoder Chip," *in Proceedings of IEEE International Solid-State Circuits Conference*, pp.314-615, 2008.

[38] Y. Murachi, K. Mizuno, J. Miyakoshi, M. Hamamoto, T. Iinuma, T. Ishihara, Y. Fang, L. Jangchung, T. Kamino, H. Kawaguchi, and M. Yoshimoto, "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*, pp.848-851, 2008.

[39] K. Babionitakis, G. Lentaris, K. Nakos, D. Reisis, N. Vlassopoulos, G. Doumenis, G. Georgakarakos, and J. Sifnaios, "An Efficient H.264 VLSI Advanced Video Encoder," *in Proceedings of 13th IEEE International Conference on Electronics, Circuits and Systems*, pp.545-548, 2006.

[40] M. Sayed, I. Amer, and W. Badawy, "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*, 2006.

[41] C.-Y. Tsai, T.-C. Chen, T.-W. Chen, and L.-G. Chen, "Bandwidth optimized motion compensation hardware design for H.264/AVC HDTV decoder," *in Proceedings of 48th Midwest Symposium on Circuits and Systems*, pp.1199-1202 Vol. 2, 2005.

[42] C.-H. Chang, J.-W. Chen, H.-C. Chang, Y.-C. Yang, J.-S. Wang, and J.-I. Guo, "A Quality Scalable H.264/AVC Baseline Intra Encoder for High Definition Video Applicaitons," *in Proceedings of IEEE Workshop on Signal Processing Systems*, pp.521-526, 2007.

[43] C.-H. Tsai, Y.-W. Huang, and L.-G. Chen, "Algorithm and architecture optimization for full-mode encoding of H.264/AVC intra prediction," *in Proceedings of 48th Midwest Symposium on Circuits and Systems*, pp.47-50 Vol. 1, 2005.

[44] A. Moffat and A. Turpin, *Compression and Coding Algorithms*, Vol. Dordrecht, the Netherlands, Kluwer Academic Publishers, 2002.

[45] D.A. Huffman, "A method for the construction of minimum redundancy codes," *Proceedings of IRE*, vol.40, no.10, pp. 1098-1101, 1952.

[46] C.E. Shannon, "A Mathematical theory of communication," *The Bell System Technical Journal*, vol.27, pp. 379-423, 623-656, 1948.

[47] N. Abramson, *Information theory and coding*, Vol. New York, McGraw-Hill Book Co., Inc., 1963.

[48] J.J. Rissanen, "Generalized kraft Inequality and arithmetic coding," *IBM Journal of Research and Development*, vol.20, no.198, pp. 1976.

[49] R.C. Pasco, *Source coding algorithms for fast data compression*, in *Department of Electrical Engineering*. 1976, Standford University.

[50] G.G. Langdon, "An Introduction to Arithmetic Coding," *IBM Journal of Research and Development*, vol.28, pp. 135-149, 1984.

[51] I.H. Witten, R.M. Neal, and J.G. Cleary, "Arithmetic Coding for Data Compression," *Communications of the ACM*, vol.30, no.6, pp. 520-540, 1987.

[52] W.B. Pennebaker, J.L. Mitchell, J. G. G. Langdon, and R.B. Arps, "An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder," *IBM Journal of Research and Development*, vol.32, no.6, pp. 717-726, 1988.

[53] J. Mitchell and W. Pennebaker, *JPEG: Still Image Data Compression Standard*, Vol. Van Nostrand Reinhold, 1993.

[54] D.S. Taubman and M.W. Marcellin, *JPEG2000 image compression fundamentals, standards and practice*, Vol. Kluwer Academic Publishers, 2002.

[55] D. Marpe, G. Blättermann, and T. Wiegand, "Adaptive Codes for H.26L," ITU-T SG16/Q.6 Doc. VCEG-L13, Eibsee, Germany, 2001.

[56] D. Marpe, G. Blättermann, G. Heising, and T.Wiegand, "Further Results for CABAC Entropy Coding Scheme," ITU-T SG16/Q.6 Doc. VCEG-M59, Austin, TX, USA, 2001.

[57] D. Marpe, G. Blättermann, and T. Wiegand, "Improved CABAC," ITU-T SG16/Q.6 Doc. VCEG-O18, Pattaya, Thailand, 2001.

[58] D. Marpe and H.L. Cycon, "Very low bit-rate video coding using wavelet-based techniques," *IEEE Transaction on Circuits System for Video Technology*, vol.9, no.4, pp. 85-94, 1999.

[59] G. Heising, D. Marpe, H.L. Cycon, and A.P. Petukhov, "Wavelet-Based very low bit-rate video coding using image warping and overlapped block motion compensation," *IEE Proceedings Vision, Image & Signal Processing*, vol.148, no.2, pp. 93–101, 2001.

[60] J. Teuhola, "A Compression Method for Clustered Bit-Vectors," *Information Processing Letters*, vol.7, no.10, pp. 308-311, 1978.

[61] S. Golomb, "Run-length encodings," *IEEE Transactions on Information Theory*, vol.12, no.3, pp. 399-401, 1966.

[62] D. Marpe and H.L. Cycon, "Efficient pre-coding techniques forwaveletbased image compression," *in Proceedings of Picture Coding Symposuim* pp.45–50, 1997.

[63] M. Mrak, D. Marpe, and T. Wiegand, "A context modeling algorithm and its application in video compression," *in Proceedings of International Conference on Image Processing*, pp.III-845-848 vol.2, 2003.

[64] K. Muller, A. Smolic, M. Kautzner, P. Eisert, and T. Wiegand, "Predictive compression of dynamic 3D meshes," *in Proceedings of IEEE International Conference on Image Processing*, pp.I-621-624, 2005.

[65] V. Sanchez, P. Nasiopoulos, and R. Abugharbieh, "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, 2008.

[66] L. Zhang, X. Wu, N. Zhang, W. Gao, Q. Wang, and D. Zhao, "Context-based Arithmetic Coding Reexamined for DCT Video Compression," *in Proceedings of IEEE International Symposium on Circuits and Systems*, pp.3147-3150, 2007.

[67] Y. Wu and J.W. Woods, "Scalable Motion Vector Coding Based on CABAC for MC-EZBC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.17, no.6, pp. 790-795, 2007.

[68] Y. Sehoon and A. Vetro, "RD-Optimized View Synthesis Prediction for Multiview Video Coding," *in Proceedings of IEEE International Conference on Image Processing*, pp.I - 209-212, 2007.

[69] R.C. Kordasiewicz, M.D. Gallant, and S. Shirani, "Encoding of Affine Motion Vectors," *IEEE Transactions on Multimedia*, vol.9, no.7, pp. 1346-1356, 2007.

[70] A. Golwelkar and J.W. Woods, "Motion-Compensated Temporal Filtering and Motion Vector Coding Using Biorthogonal Filters," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.17, no.4, pp. 417-428, 2007.

[71] C. Sun, H.-J. Wang, H. Li, T.-H. Kim, and X.-B. Yu, "An Efficient Context Modeling Algorithm for Motion Vectors in CABAC," *in Proceedings of IEEE International Symposium on Signal Processing and Information Technology*, pp.796-800, 2007.

[72] J.H. Lin and K.K. Parhi, "Parallelization of Context-Based Adaptive Binary Arithmetic Coders," *IEEE Transactions on Signal Processing*, vol.54, no.10, pp. 3702-3711, 2006.

[73] D. Levine, W.E. Lynch, and L.-N. Tho, "Observations on error detection in H.264," *in Proceedings of 50th Midwest Symposium on Circuits and Systems*, pp.815-818, 2007.

[74] Y. Li, H. Xiong, L. Song, and S. Yu, "A Context-Based Error Detection Strategy into H.264/AVC CABAC," *in Proceedings of IEEE International Conference on Multimedia and Expo*, pp.689-692, 2006.

[75] Y. Wang and S. Yu, "Joint source-channel decoding for H.264 coded video stream," *IEEE Transactions on Consumer Electronics*, vol.51, no.4, pp. 1273-1276, 2005.

[76] S.B. Jamaa, M. Kieffer, and P. Duhamel, "Controlled Complexity Map Decoding of CABAC Encoded Data," *in Proceedings of IEEE International Conference on Multimedia and Expo*, pp.1441-1444, 2006.

[77] W. Yu and Y. He, "A high performance CABAC decoding architecture," *IEEE Transactions on Consumer Electronics*, vol.51, no.4, pp. 1352-1359, 2005.

[78] B. Li, D. Zhang, J. Fang, L. Wang, and M. Zhang, "A high-performance VLSI architecture for CABAC decoding in H.264/AVC," *in Proceedings of 7th International Conference on ASIC*, pp.790-793, 2007.

[79] J.-W. Chen and Y.-L. Lin, "A High-Performance Hardwired CABAC Decoder," *in Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pp.II-37-II-40, 2007.

[80] C.-H. Kim and I.-C. Park, "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*, 2006.

[81] Y. Yi and I.C. Park, "High-Speed H.264/AVC CABAC Decoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.17, no.4, pp. 490-494, 2007.

[82] J.-W. Chen, C.-R. Chang, and Y.-L. Lin, "A hardware accelerator for context-based adaptive binary arithmetic decoding in H.264/AVC," *in Proceedings of IEEE International Symposium on Circuits and Systems*, pp.4525-4528 Vol. 5, 2005.

[83] H. Eeckhaut, M. Christiaens, D. Stroobandt, and V. Nollet, "Optimizing the critical loop in the H.264/AVC CABAC decoder," *in Proceedings of IEEE International Conference on Field Programmable Technology*, pp.113-118, 2006.

[84] B. Shi, W. Zheng, H.-S. Lee, D.-X. Li, and M. Zhang, "Pipelined Architecture Design of H.264/AVC CABAC Real-Time Decoding," *in Proceedings of 4th IEEE International Conference on Circuits and Systems for Communications*, pp.492-496, 2008.

[85] W. Son and I.-C. Park, "Prediction-based real-time CABAC decoder for high definition H.264/AVC," *in Proceedings of IEEE International Symposium on Circuits and Systems*, pp.33-36, 2008.

[86] L. Li, Y. Song, T. Ikenaga, and S. Goto, "A CABAC Encoding Core with Dynamic Pipeline for H.264/AVC Main Profile," *in Proceedings of IEEE Asia Pacific Conference on Circuits and Systems*, pp.760-763, 2006.

[87] C.-C. Kuo and S.-F. Lei, "Design of a Low Power Architecture for CABAC Encoder in H.264," *in Proceedings of IEEE Asia Pacific Conference on Circuits and Systems*, pp.243-246, 2006.

[88] O. Flordal, D. Wu, and D. Liu, "Accelerating CABAC encoding for multi-standard media with configurability," *in Proceedings of 20th International Parallel and Distributed Processing Symposium*, 2006.

[89] J.-L. Chen, Y.-K. Lin, and T.-S. Chang, "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*, pp.II-105-108, 2007.

[90] P.-S. Liu, J.-W. Chen, and Y.-L. Lin, "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*, pp.1-4, 2007.

[91] C.-C. Lo, Y.-J. Zeng, and M.-D. Shieh, "Design and test of a highthroughput cabac encoder," *in Proceedings of IEEE Region 10 Conference*, pp.1-4, 2007.

[92] Y.-J. Chen, C.-H. Tsai, and L.-G. Chen, "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*, pp.1-4, 2006.

[93] J.L. Nunez-Yanez, V.A. Chouliaras, D. Alfonso, and F.S. Rovati, "Hardware assisted rate distortion optimization with embedded CABAC accelerator for the H.264 advanced video codec," *IEEE Transactions on Consumer Electronics*, vol.52, no.2, pp. 590-597, 2006.

[94] R.R. Osorio and J.D. Bruguera, "Arithmetic coding architecture for H.264/AVC CABAC compression system," *in Proceedings of Euromicro Symposium on Digital System Design*, pp.62-69, 2004.

[95] R.R. Osorio and J.D. Bruguera, "High-Throughput Architecture for H.264/AVC CABAC Compression System," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.16, no.11, pp. 1376-1384, 2006.

[96] T.M. Le, X.H. Tian, B.L. Ho, J. Nankoo, and Y. Lian, "System-on-Chip Design Methodology for a Statistical Coder," *in Proceedings of Seventeenth IEEE International Workshop on Rapid System Prototyping*, pp.82-90, 2006.

[97] *Reference codec software: JM 12.4*, Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Available from: http://iphome.hhi.de/suehring/tml/

[98] *PIN software analysis tool*, University of Colorado, Available from: http://rogue.colorado.edu/pin/

[99] J.L. Nunez-Yanez, V.A. Chouliaras, and D. Alfonso, "Hardware assisted rate distortion optimization with embedded CABAC accelerator for the H.264 advanced video codec," *in Proceedings of International Conference on Consumer Electronics*, pp.95-96, 2006.

[100] M. Li and W. Wu, "A high throughput binary arithmetic coding engine for H.264/AVC," *in Proceedings of 8th International Conference on Solid-State and Integrated Circuit Technology*, pp.1914-1918, 2006.

[101] S. Sudharsanan and A. Cohen, "A hardware architecture for a context-adaptive binary arithmetic coder," *in Proceedings of SPIE Embedded processors for multimedia and communications II*, pp.104-112, 2005.

[102] X.H. Tian, T.M. Le, X. Jiang, and Y. Lian, "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, 2008.

[103] R.R. Osorio and J.D. Bruguera, "A new architecture for fast arithmetic coding in H.264 advanced video coder," *in Proceedings of 8th Euromicro Conference on Digital System Design*, pp.298-305, 2005.

[104] *WISHBONE System-on-a-Chip Interconnection Architecture for Portable IP Cores*, Revision B.3 Specification. Vol. OPENCORES, 2002.

[105] H. Shojania and S. Sudharsanan, "A high performance CABAC encoder," *in Proceedings of 3rd International IEEE-NEWCAS Conference*, pp.315-318, 2005.

[106] V.H.S. Ha, W.-S. Shim, and J.-W. Kim, "Real-time MPEG-4 AVC/H.264 CABAC entropy coder," *in Proceedings of International Conference on Consumer Electronics*, pp.255-256, 2005.

[107] B.L. Ho, *Performance and Complexity Analyses of H.264/AVC CABAC Entropy Coder*, in *Department of Electrical and Computer Engineering*. 2006, National University of Singapore: Singapore.

[108] *AMBA™ Specification (Rev 2.0)*, Vol. ARM, 1999.

[109] M. Weber, "Arbiters: Design Ideas and Coding Styles," *in Proceedings of SNUG of Synopsys, Boston*, 2002.

[110] *ISE Webpack Software*, Xilinx, Available from: www.xilinx.com/ise/logic_design_prod/webpack.htm

[111] *ModelSim SE 6.1b*, Mentor Graphics, Available from: http://www.model.com/downloads/default.asp

[112] *ChipScope Pro*, Xilinx, Available from: http://www.xilinx.com/ise/optional_prod/cspro.htm

[113] *Design Compiler*, Synopsys, Available from: www.synopsys.com

[114] *Astro*, Synopsys, Available from: www.synopsys.com

[115] *Power Compiler*, Synopsys, Available from: www.synopsys.com

[116] J.M. Rabaey, 2006 *Issues in Low Power Design - Minimizing Active Power*. 2006

[117] *MBISTArchitect*, Mentor Graphics, Available from: http://www.mentor.com/products/silicon-yield/memorytest/mbistarchitect/

[118] *MicroBlaze Processor*, Xilinx, Available from: http://www.xilinx.com/products/design_resources/proc_central/microblaze.htm