

# Implementation Strategies for Statistical Codec Designs in H.264/AVC Standard

X.H. Tian, T.M. Le *SM-IEEE*, X. Jiang, Y. Lian *SM-IEEE*  
*Department of Electrical and Computer Engineering*  
*National University of Singapore*  
*elelmt@nus.edu.sg*

## Abstract

*Two statistical coding tools - the Context-based Adaptive Variable Length Coding (CAVLC) and the Context-based Adaptive Binary Arithmetic Coding (CABAC) - have been adopted in different profiles of the H.264/AVC video coding standard. The throughput at the statistical coding stage is mainly constrained by the high data dependency and sequential coding nature of CAVLC and CABAC. Many hardware designs have been proposed to remove the bottlenecks and accelerate statistical coding and decoding of H.264/AVC. In this paper, different implementation strategies of CAVLC and CABAC encoder and decoder architectures are investigated. The strategies are evaluated using criteria such as circuit area, processing time, and power consumption. The three most important techniques used are: multi-symbol processing, table lookup optimization, and critical path reduction by data prefetch and pre-calculation. In the discussion of CABAC encoder design, our implementation strategies are introduced and compared with other reported designs.*

## 1 Introduction

The newest international video coding standard H.264/AVC [1-2] has significantly improved video compression performance and its network friendliness, compared to the existing standards such as MPEG-2 [3], and H.263 [4]. A number of new coding techniques are adopted in H.264/AVC, including variable block-size motion compensation, spatial prediction for intra coding, in-loop de-blocking filtering, support for small block-size (4×4) transform, Rate-Distortion Optimization (RDO) [5], context adaptive entropy (statistical) coding, etc. Statistical coding is used to further remove the redundancy of code words after spatial and temporal redundancy reductions. Two statistical coding tools: the Context-based Adaptive Variable Length Coding (CAVLC) [6] and the Context-based Adaptive Binary Arithmetic Coding (CABAC) [7] have been adopted in different profiles of the H.264/AVC video coding standard. In the Baseline and Extended profiles, CAVLC is utilized, and in the Main and High

profiles targeting high bit rate high definition service, CABAC is used.

Recognizing the highly computational complexity of motion estimation, because of the serial coding nature and high data dependency of coding procedure in both CAVLC and CABAC, the throughput of a video codec is also limited by the statistical coding stage. As it is not efficient to remove the bottleneck by software optimization and acceleration alone, several hardware designs for CAVLC and CABAC have been proposed [8-31], to enhance throughput in various applications. In this paper, different implementation strategies of CAVLC and CABAC encoding and decoding architectures will be investigated. The strategies are evaluated using circuit area, processing time, and power consumption as judging criteria. They 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. In the following sections, CAVLC and CABAC codec implementation strategies will be discussed in sections 2 and 3, respectively. Also in section 3, our CABAC encoder design is discussed and compared with the related designs. Conclusions will be given in the final section.

## 2 CAVLC codec implementation strategies

In the Baseline and Extended profiles of H.264/AVC, two statistical coding methods are supported: Exp-Golomb coding and CAVLC [2]. The coding and decoding procedures of Exp-Golomb coding are simple and regular, but the coding efficiency is low. To encode the SEs of quantized transformed coefficients, a more efficient CAVLC scheme is applied. The transformed coefficients of the 4×4 non-DC block or 2×2 DC block are scanned in a zig-zag pattern and represented by data pairs of run and level. Because run and level are not quite correlated, they are mapped into five types of SEs, and processed separately. The five SE types include:

- (1) `Coeff_token`: the number of non-zero coefficients and trailing 1s (T1s) in the block. T1 indicates the number of coefficients with absolute value 1 at the

end of the zig-zag scan. One of three VLC tables or one fixed-length-table is selected to encode coeff\_token based on the number of non-zero coefficients in the neighboring coded blocks.

- (2) Sign\_T1s: 1-bit SE indicates the sign of the T1 coefficients. The length of T1s can be from 0 to 3.
- (3) Level: value of the coefficient level with absolute value greater than 1, represented by the level prefix and suffix string. Coefficients of T1s are represented by the two SEs discussed above.
- (4) Total\_zeros: the total number of zero coefficients of one block, preceding the last non-zero coefficient in zig-zag scan order.
- (5) Run\_before: the number of successive zero coefficients before each non-zero coefficient in zig-zag scan order.

CAVLC decoding and encoding procedures and implementation strategies are discussed in the following two sub-sections.

## 2.1 CAVLC decoder design

### 2.1.1 CAVLC decoding procedure

The CAVLC decoding flow is shown in Fig. 1. It consists of 6 stages: (1) coeff\_token is parsed from the input bit stream by a table lookup, and the number of non-zero coefficients and T1s are obtained; (2) sign\_T1s are parsed to get the signs of T1s and reconstruct T1 values of the residual block; (3) prefix and suffix of level are parsed to calculate level values. All level values are collected and buffered; (4) total\_zeros is parsed; (5) run\_before values of the block are recursively parsed; and (6) all coefficients of the block are reconstructed according to {run, level} data pairs in reverse zig-zag scan order. The reconstructed coefficient block is utilized by the inverse quantization (IQ) and reverse transform stage of H.264/AVC decoder.

### 2.1.2 CAVLC decoder implementation strategies

According to the CAVLC decoding flow, implementation strategies of each stage are illustrated in Fig. 1. In [8], short codewords (high probability) are parsed by a simple combinational circuit, while remaining long codewords (low probability) are parsed by accessing the lookup tables (LUT) from memory. Therefore, large ratio (65-88%) of memory access was reduced in [8]. However, LUT can be more efficiently implemented using fully combinational circuit, which is efficient in both circuit area and processing time, because memory space of LUT is saved and the memory access time is removed.

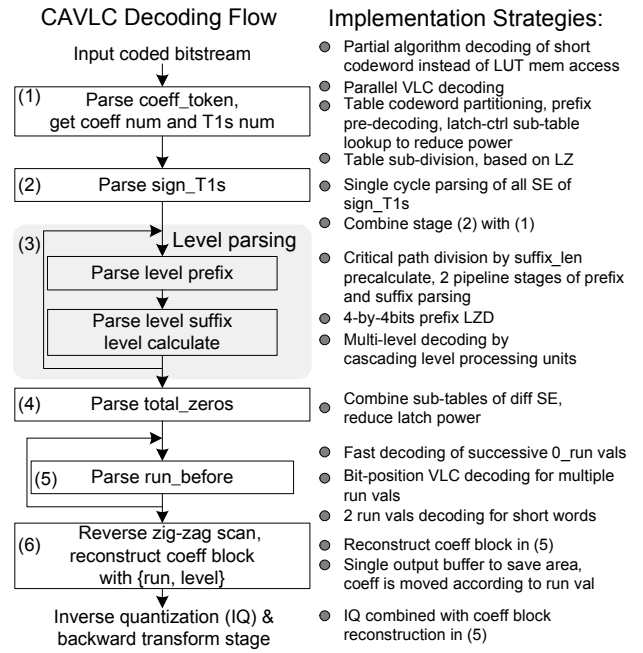


Figure 1: CAVLC decoding flow and implementation strategies utilized in the decoding stages

In stage (1), LUT can be redesigned, and long codeword of original LUT can be partitioned to prefix and suffix [9]. If the parsing codeword is not in the prefix LUT, it is further looked up in the suffix table. Because of relatively low probabilities of long codewords, and smaller prefix LUT, the average table lookup operation is faster. Latches are allocated to control the table lookup procedure of both prefix and suffix LUTs during codeword parsing, and unnecessary table access is avoided and 40% of power consumption is reduced. The strategy is useful for the design that implements LUT by combinational circuit. Similar methods of LUT redesign are reported in [10] and [11], in which original LUTs are divided into sub-tables based on the number of leading zeros (LZ) in the codeword. Sub-table selection is implemented by fast LZ detection (LZD) circuit. The benefits of these methods are: smaller circuit area of LUT and shorter parsing time.

In stage (2), parsing of sign\_T1s can be implemented in a single cycle [9], [10], as it is fixed-length SE parsing, and the number of T1s is already obtained in stage (1). In [11] and [13], this stage is combined with stage (1) to reduce the one cycle processing time per block.

In stage (3), suffix\_length of the current level is updated after parsing of level suffix, and it is required by the parsing procedure of next level. In [9], this data dependency is removed by the pre-calculation of next suffix\_length in the prefix parsing phase. So the level parsing can be divided into 2 pipeline stages with shorter

critical path. Because the prefix of level is parsed by LZD, a dedicated 4-by-4bit LZD circuit is proposed in [10] to accelerate LZD procedure. Accordingly, the critical path of level decoding can also be reduced. Dual-level decoding circuit is proposed in [13] by cascading two levels of decoding units. However, as data dependency of suffix\_length is not removed in [13], its implementation is not as efficient as [9].

In stage (4), sub-tables of total\_zero are combined with similar sub-tables of coeff\_token [9]. Because stage (1) and (4) do not occur simultaneously, the number of LUTs is reduced, and the circuit area and power consumption of gating latches of LUT are also reduced. The drawback is that table lookup time is prolonged.

In stage (5), because the codeword of run\_before LUT is short (3 bits or less), 2 values of run\_before are decoding per cycle in [10] with minimum addition to circuit area. When run\_before is 0, the codeword contains only bit 1s. In the high bit rate coding, successive 0s of run\_before frequently occur, which can be quickly parsed by counting the length of successive 1s of input bit string [13]. But the circuit area of this method is larger than that of [10], and the benefit is less significant in low bit rate applications. Bit-position VLC decoding for multiple run\_before values in [12] can also accelerate run\_before parsing. However, the circuit area is significantly larger, and critical path is longer.

Reconstruction of coefficient block of stage (6) is combined with stage (5) in [9] and [14]. This is because when run\_before is known, the position of coefficient can be decided. Single output buffer is allocated in [9] to store non-zero coefficient levels in stage (3). The coefficients in the buffer are moved to the correct position of coefficient block in stage (5). This strategy saves both an additional buffer for block reconstruction and the processing time of reconstruction. IQ operation is combined with the reconstruction procedure of coefficient block in the 2-stage pipeline architecture in [14] to save the processing time of IQ. The strategies discussed in [9] and [14] are adoptable to design the interface between CAVLC decoding and following stages of IQ and backward transform.

## 2.2 CAVLC encoder design

### 2.2.1 CAVLC encoding procedure

The CAVLC encoding flow is shown in Fig. 2. It consists of 6 coding stages, including: (1) zig-zag scan of 4×4 or 2×2 quantized transform block to generate SE values of coeff\_token, sign\_T1s, level, total\_zeros, and run\_before; (2) encode coeff\_token by LUT; (3) encode sign\_T1s; (4) recursive encode non-T1 levels in reverse zig-zag scan order, output prefix and suffix bit string of level; (5) encode total\_zeros by LUT; (6) recursive encode

each run\_before of non-zero coefficient of the block in the reverse zig-zag scan order. CAVLC encoding is the reverse procedure of decoding. However, because all transform coefficients of the block are available before encoding, the data dependency is lower, compared to decoding. Therefore, it is more feasible to design fast coding circuits.

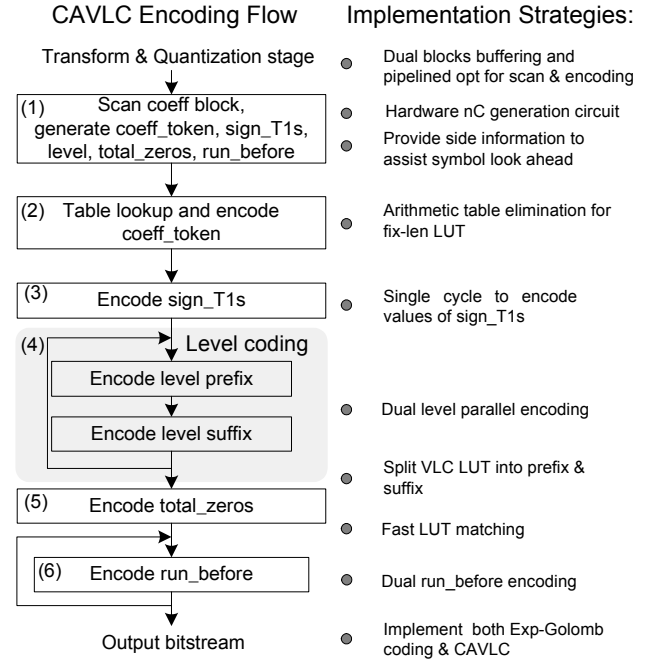


Figure 2: CAVLC encoding flow and implementation strategies utilized in the encoding stages

### 2.2.2 CAVLC encoder implementation strategies

Several implementation strategies of CAVLC encoding are listed in Fig. 2. To achieve complete statistical coding functions, Exp-Golomb coding circuits are integrated with the CAVLC encoder in [15] and [16]. Because the coefficient scan of stage (1) is time-consuming, dual SE buffers are allocated in [17] to enable a two-stage pipelined encoding. The scan stage scans the coefficient block and buffers the SE values into one buffer, while the encoding stage accesses the other buffer to fetch and encode SE values. However, the additional buffer increases the circuit area and power consumption. In low bit-rate coding, the number of non-zero coefficients is small. Thus, scan cycles can be longer than the encoding cycles, and pipeline efficiency can be degraded.

Another method: side information aided (SIA) symbol look ahead (SLA) is proposed in [16] to reduce the processing time of the scan procedure. The map of non-zero coefficients and coefficients with absolute value 1 are provided to assist direct generation of CAVLC SEs without a scan procedure. Compared to [17], the circuit area and

power consumption of SE buffers are removed. The drawback of this scheme is that the side information maps still needs to be calculated by the host processor, which entails a prolonged processing time, and limits the actual CAVLC coding performance. A better scheme proposed in this paper is to utilize parallel hardware comparators to generate a map of non-zero and abs\_1 coefficient in single cycle. As the circuit area increase in parallel comparators is insignificant, the net result is a faster CAVLC coding and lower host computation.

In stage (1), the estimated coefficient number ( $n_C$ ) is calculated to select LUT of coeff\_token. The calculation of  $n_C$  needs to utilize the coefficient number of neighboring coded block, and coefficient number of current coded block needs to be saved. Only [15] generates  $n_C$  in hardware design with the benefits of more complete hardware function of CAVLC and lower memory access frequency on the host processor, although the circuit area is larger.

In stage (2), to reduce the LUT circuit area (5% reduction), VLC tables are split in [17], and SEs are mapped to prefix and suffix codeword before final concatenation. In [18], arithmetic table elimination is adopted to replace the lookup of fixed-length LUT of coeff\_token by simple arithmetic coding circuit, with minimum increase in circuit area.

In stage (3), values of sign\_T1s are also coded in single cycle [18], similar to the decoding strategy. In stage (4), because all level values are available, data dependency of suffix\_length is not critical during encoding. Two processing units of level are allocated [19], and the suffix\_length generated in the 1<sup>st</sup> unit is sent to the 2<sup>nd</sup> unit to encode the 2<sup>nd</sup> level. In stage (6) of [19], dual run\_before coding is also supported by arithmetic coding instead of LUT. The block coding cycles are significantly reduced with such multi-SE encoding strategy of [19].

### 3 CABAC codec implementation strategies

CABAC achieves an average of 9-14% savings in bit-rate over CAVLC [6], because arithmetic coding can represent each coding symbol (bin) of alphabet with non-integer number of bits, and adaptive update of context (probability) model of coding symbol based on coded SEs can more precisely estimate and represent the probability of the value of coding symbol. In H.264/AVC encoding, RDO also contributes an average of over 13% bit-rate savings in CIF and SDTV tests [20]. The combination of RDO and CABAC achieves around 20% bit-rate savings. However, the encoding computation complexity is significantly increased, occupying around 1/3 of total encoder instructions running on host processor.

CABAC encoder consists of 3 functional components: (1) binarization to map SE value to bin string; (2) context model selection to select the proper context model for regular coding 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 Range (R) and selecting one of the two sub-intervals  $R_{LPS}$  and  $R_{MPS}$  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 R are shifted out as coding result when the bin values are fixed. CtxIdx is the sum of context offset (CtxOffset) and context index increment (CtxIdxInc). CtxOffset is decided by SE type, while CtxIdxInc depends on values of coded bin or coded SE in the neighboring coded MB.

The following sub-sections 3.1 and 3.2 introduce the proposed implementation strategies applied to the decoding and encoding flow of CABAC recently reported in the literatures, and analyze the benefits and limitations of the strategies. Our CABAC encoder implementation strategies are also discussed in 3.2.

#### 3.1 CABAC decoder design

Block diagram of CABAC decoder of H.264/AVC is illustrated in Fig. 3, including the following 3 function stages: (1) binary arithmetic decoding, (2) context modeler selection, and (3) binarization matching. The decoding flow shown in the figure is the reverse procedure of CABAC encoding flow.

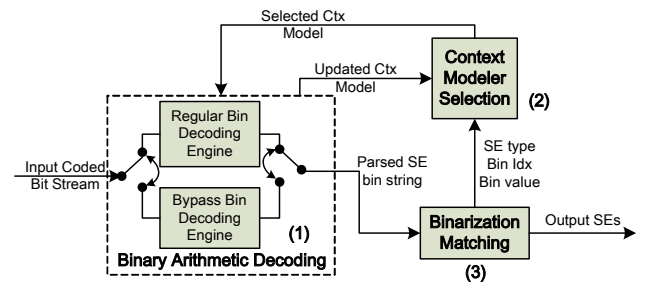


Figure 3: Block diagram of CABAC decoder

##### 3.1.1 Binary arithmetic decoding (BAD)

Coding bins of RB and BB are parsed separately by two decoding engines. Context model is provided by stage (2) 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 the data dependency, it is not easy to achieve multi-bin decoding per cycle in such situation. However, when the context access pattern is fixed, such as decoding the residual SE including significant coefficient flag (SCF), last SCF (LSCF), and level,

cascaded decoding of multi-bin can be achieved to decode 2 RB, 2BB, or 1 RB and 1BB per cycle [21], [22], [23]. To reduce the critical path of dual RB decoding, two possible  $R_{LPS\_4}$  values of the 2<sup>nd</sup> RB are pre-calculated during 1<sup>st</sup> bin decoding [21], and the correct  $R_{LPS\_4}$  value is selected for the 2<sup>nd</sup> bin when 1<sup>st</sup> bin is decoded. Dual RB decoding of [24] constantly predicts that 1<sup>st</sup> RB is MPS, and begins the 2<sup>nd</sup> RB decoding. Critical path of RB decoding is shorter than [21]; however, the throughput of [24] is only 0.56 bin/cycle, as for the wrong prediction, the 2<sup>nd</sup> decoded bin is discarded.

To achieve even higher throughput, line-bit-rate strategy is proposed in [25] to accelerate decoding of residual SE. Sixteen RB decoding units are cascaded, and the parsed bin of one stage is used to calculate the CtxIdx of the next bin. Groups of context models of the residual SE are stored in register file to allow direct access of context models by each RB decoding unit. Although throughput can be enhanced for residual SE decoding, the critical path is significantly longer, and it does not improve the decoding throughput of other SE types. The average throughput of [25] is only 1 bin/cycle, while circuit area and power consumption of large context register files and BAD circuits prevents the scheme from being implemented in lower power applications.

### 3.1.2 Context model selection

Context model of the next RB is selected in this stage by calculating CtxIdx based the coding SE type, bin index, and current decoded bin. In order to reduce context RAM access delay, a group of context models can be prefetched to local buffers [21], [26]. For the context model of coded block pattern (CBF), CtxIdx of the next CBF is pre-calculated during decoding of current block [23]. Context models of LSCF are stored in a separate SRAM to enable simultaneously context access for SE pair of SCF and LSCF and faster decoding of SCF and LSCF [23]. The techniques of context model prefetch, CtxIdx pre-calculation, and parallel access of multiple context models are beneficial to reduce context access delay and processing time.

### 3.1.3 Binarization match, and SE generation

Binarization matching is the reverse procedure of binarization of CABAC encoding. It is controlled by FSM to identify the type of next decoding SE. The corresponding LUT of the SE is accessed to match the decoded codeword. The parsed SE is output from this stage. FSM optimization and combinational circuit implementation of simple LUT can be adopted to accelerate the matching procedure.

## 3.2 CABAC encoder design

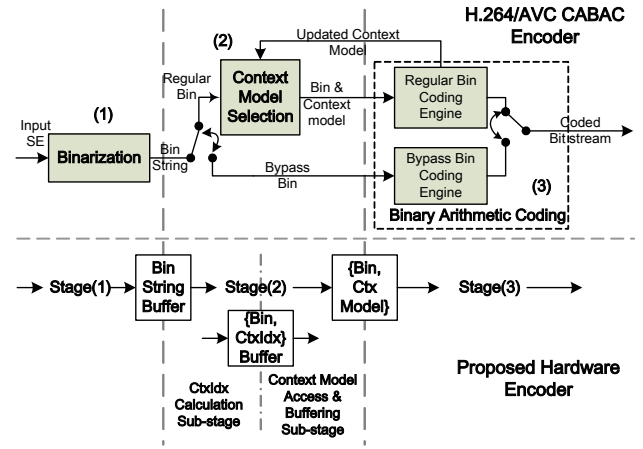


Figure 4: Block diagram of CABAC encoder [4], and diagram of the proposed HW CABAC encoder

CABAC encoder block diagram of H.264/AVC is shown in the top part of Fig. 4. As introduced previously, the encoder consists of 3 major functional stages. In CABAC encoding procedure, the data dependency is not high between stage (2) context model selection and stage (3) BAC, compared to the decoding flow. It is because after binarization, the bin string of SE is available to calculate the CtxIdxInc and CtxIdx of each RB. Generated CtxIdx can be buffered to access the context model from SRAM. Coding stages of our proposed CABAC encoder is illustrated at the bottom part of Fig. 4. It is also partitioned into 3 function stages. Stage (2) is further divided to 2 sub-stages: CtxIdx calculation, and context model access and buffering (CA).

With the 2 pipeline buffers inserted (Fig. 4), throughput of 1 bin/cycle is achieved, because 3 stages of this encoder design can simultaneously work. The design is synthesized using AMS 0.35  $\mu$ m process and Xilinx Virtex4 technology. The circuit synthesis results are shown in Table I, and compared with related recent CABAC encoder designs in the aspect of throughput, maximum clock frequency, and circuit area. The throughput of our design is faster than that of [27], [28], and [29] (over 1.5 times speed up of throughput), because in our design, function of context access and BAC is properly partitioned into 3 pipeline stages with data prefetch and pre-calculation techniques utilized. The address of context model is pre-calculated and context model is prefetched, and the risk of pipeline bubble is removed, compared to [29]; and bypass path of context RAM access is enabled in stage (2) to avoid the conflicts of RAM read and write operations of [28] when the same context model is successively accessed. The throughput of [27] is low, because fully hardware implementation of CtxIdxInc calculation takes long period to access SE of

neighboring coded MB. In our previous work [30], all CtxIdxInc values of RB are generated by host processor, which cause high computation complexity of host processor and degradation of system performance when CtxIdxInc can not be calculated in time. In this paper, most CtxIdxInc values are generated in stage (1) and (2) of the CABAC encoder. Only when the SEs of neighboring coded MBs are used to calculate CtxIdxInc, it is calculated by the host processor. Compared to [27], this SW/HW partition strategy avoids the unnecessary increase in HW complexity and high memory resource requirement of storing SEs of coded MBs in hardware. In addition, host processor computation for this part of CtxIdxInc values is low, average of 12% of total number of coding bins.

**Table I. Circuits synthesize results of CABAC encoder designs**

Design	Process technology	Throughput Bin/Cycle	Clock Freq MHz	Circuit area of logic functions
[20]	Xilinx Virtex4	1	130	856 slices (CA+stage3)
[27]	0.13 $\mu\text{m}$	0.67	200	34.3K gates
[28]	0.15 $\mu\text{m}$	0.56	333	13.3K (no CtxIdxInc calc)
[29]	0.35 $\mu\text{m}$	0.59	150	4.57K gates (CA+stage3)
[31]	0.35 $\mu\text{m}$	1.9~2.3	186	19.4K gates
This paper	Xilinx Virtex4	1	137	2746 slices (total) 941 slices (CA+stage3)
	0.35 $\mu\text{m}$		186	19.1K gates

Throughput of [20] is also 1 bin/cycle. Only [20] and our design fully support context state backup and restoration operation of RDO coding modes, which is necessary in high quality video coding applications. In our design, context models in the SRAM are accessed by context line (8 context model) and buffered in two context line buffers of stage (2). And the context RAM is remapped to allocate the context models of same SE type in the same or neighboring context lines. With these strategies, the average RAM access frequency of this design is 28.4% of [20] in the non-RDO coding mode, and it is significantly lower. Strategy of pipelined context state backup and restoration operations is proposed in our design for P8 $\times$ 8 RDO coding, and the average operation delay for P and B frames is 15.5% and 16.6% of [20], which is also significantly lower than [20]. In [31], two RBs can be processed per cycle, with higher throughput (1.9-2.3 bin/cycle, and 1.5 times speed up of bin/second). Two RB processing units of coding interval R are cascaded to update R for two RBs per cycle. However, the design only focuses on HW acceleration of residual SEs. The RB or BB data pairs and related CtxIdxInc and CtxIdx values of non-residual SEs and CBF are generated by host processors. So the host processor computation complexity of [31] is significantly higher than ours. The HW encoder of [31] and our design achieve similar circuit area ([31]: 19.4K gates, our design: 19.1K gates) at 186MHz clock using the same 0.35 $\mu\text{m}$  process. If power consumption of both HW

encoder and host processor are considered, the power consumption of [31] is in theory higher than ours.

The circuit area of this design is larger than [20], [28], and [29], because more functional stages are implemented in this design including binarization, CtxIdxInc calculation, and RDO context operations. In our future work, the design can be extended to the full hardware calculation of CtxIdxInc with efficient SRAM backup and restoration operation for the coded SEs. With limited increase of circuit area and processing time, the function completeness and integratability of the design can be further improved.

## 4 Conclusions

In this paper, hardware design implementation strategies are investigated on the statistical coding tools CAVLC and CABAC of H.264/AVC standard. The implementation strategies utilized in CAVLC decoder and encoder, CABAC decoder and encoder designs are introduced and discussed. The strategies are evaluated using the criteria of circuit area, processing time, and power consumption.

In the CAVLC codec designs, the strategies focus on accelerating VLC table lookup, reducing LUT circuit area and access power consumption, simplifying processing of coefficient level and zig-zag scan in encoder and coefficient block reconstruction in decoder. In the CABAC decoder designs, cascaded processing units and data pre-calculation method are utilized to accelerate decoding procedure. Context model prefetch and local buffering, separating context RAM tables for multi-context models accessing, and CtxIdxInc pre-calculation are practical to reduce context model access delay. In the CABAC encoder design, our implementation strategies are discussed and compared with related designs in terms of throughput, frequency of memory access, operation delay of context state backup and restoration in RDO coding, and power consumption.

Generally speaking, the three most important strategies utilized are: multi-symbol processing (1.5 times speed up of bin/second [31], compared to single-symbol processing), table lookup optimization (40% power reduction [9], and 65-88% memory access reduction [8]), and critical path reduction by data prefetch and pre-calculation (over 1.5 times speed up of throughput, our design compared to [27-29]). Functional completeness and processing time reduction are the two high-priority targets of many designs, while reducing power consumption is another consideration of some designs. Useful implementation strategies summarized in this paper will be referenced in our future work in statistical coder designs.

## Acknowledgment

This work is supported by the Faculty Research Committee grant (263-000-405-112 and 263-000-405-133), Faculty of Engineering, National University of Singapore.

## References

- [1] "Draft ITU-T Recommendation H.264 and Draft ISO/IEC 14496-10 AVC," in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-G050, T. Wieg, Ed., Pattaya, Thailand, Mar. 2003.
- [2] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 13, No. 7, pp. 560 – 576, July 2003.
- [3] "Information Technology-Generic Coding of Moving Pictures and Associated Audio Information: Video," 13818-2-ITU-T Rec. H.262 (MPEG-2 Video), ISO/IEC Std. 1995.
- [4] "Video Coding for Low Bitrate Communication," ITU-T Recommendation H.263, Sep. 1997.
- [5] G. J. Sullivan and T. Wiegand, "Rate distortion optimization for video compression," *IEEE Signal Processing Magazine*, pp. 74-90, Nov. 1998.
- [6] G. Bjntegaard and K. Lillevold, "Context-adaptive VLC (CAVLC) coding of coefficients", JVT-C028, 3rd Meeting: Fairfax, Virginia, USA, 6-10 May 2002.
- [7] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard", *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 13, No. 7, pp.620 – 636, July 2003.
- [8] Y. Moon, G. Kim, and J. Kim, "An efficient decoding of CAVLC in H.264/AVC video coding standard," *IEEE Trans. Consumer Electronics*, Vol. 51, No. 3, pp. 933-938, Aug. 2005.
- [9] H. Lin, Y. Lu, B. Liu, and J. Yang, "A Highly Efficient VLSI Architecture for H.264/AVC CAVLC Decoder," *IEEE Trans. Multimedia*, Vol. 10, No 1, pp.31-42, Jan. 2008
- [10] M. Alle, J. Biswas, and S.K. Nandy, "High Performance VLSI Architecture Design for H.264 CAVLC Decoder," In *Proc. Int. Conf. Application-specific Systems, Architectures and Processors*, pp. 317–322, 2006.
- [11] H. Chang, C. Lin, and J. Guo, "A novel low-cost high-performance VLSI architecture for MPEG-4 AVC/H.264 CAVLC decoding," In *Proc. ISCAS 2005*, pp. 6110-6113.
- [12] Y. Wen, G. Wu, S. Chen, and Y. Hu, "Multiple-Symbol Parallel CAVLC Decoder for H.264/AVC," In *Proc. APCCAS 2006*, pp. 1240-1243, 2006.
- [13] T. Tsa, D. Fang, and Y. Pan, "A Hybrid CAVLD Architecture Design with Low Complexity and Low Power Considerations," In *Proc. IEEE Int. Conf. Multimedia and Expo*, pp. 1910-1913, 2007.
- [14] Y. Chao, S. Wei, J. Yang, and B. Liu, "Combined CAVLC Decoder and Inverse Quantizer for Efficient H.264/AVC Decoding," In *Proc. APCCAS 2006*, pp. 259-262, 2006.
- [15] D. Kim, E. Jung, H. Park, H. Shin, and D. Har, "Implementation of High Performance CAVLC for H.264/AVC Video Codec," In *Proc. 6th Int. Workshop on SOC for Real-Time Applications*, pp. 20-23, 2006.
- [16] C. Tsai, T. Chen, and L. Chen, "Low Power Entropy Coding Hardware Design for H.264/AVC Baseline Profile Encoder," In *Proc. IEEE Int. Conf. Multimedia and Expo*, pp. 1941-1944, 2006.
- [17] T. Chen, Y. Huang, C. Tsai, B. Hsieh, and L. Chen, "Architecture Design of Context-Based Adaptive Variable-Length Coding for H.264/AVC," *IEEE Trans. Circuits and Systems II*, Vol.53, No.9, pp.832-836, Sept. 2006.
- [18] C.A. Rahman and W. Badawy, "CAVLC Encoder Design for Real-Time Mobile Video Applications," *IEEE Trans. Circuits and Systems II*, Vol. 54, No. 10, pp. 873–877, Oct. 2007.
- [19] C. Chien, K. Lu, Y. Shih, and J. Guo, "A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications," *ISCAS 2006*.
- [20] 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 Trans. Consumer Electronics*, Vol. 52, No. 2, pp. 590-597, May 2006.
- [21] W. Yu and Y. He, "A high performance CABAC decoding architecture," *IEEE Trans. Consumer Electronics*, Vol. 51, No. 4, pp.1352 – 1359, Nov. 2005.
- [22] B. Li, D. Zhang, F. Jian, L. Wang, and M. Zhang, "A high-performance VLSI architecture for CABAC decoding in H.264/AVC," In *Proc. Int. Conf. ASICON*, pp.790 – 793, 2007.
- [23] J.W. Chen and Y.L. Lin, "A High-Performance Hardwired CABAC Decoder," In *Proc. ICASSP*, Vol. 2, pp.37-40, 2007.
- [24] C.H. Kim and I.C. Park, "High speed decoding of context-based adaptive binary arithmetic codes using most probable symbol prediction," In *Proc. ISCAS 2006*, May 2006.
- [25] P. Zhang, W. Gao, D. Xie, and D. Wu, "High-Performance CABAC Engine for H.264/AVC High Definition Real-Time Decoding," In *Proc. ICCE 2007*, pp.1-2, 2007.
- [26] Y. Yi and I.C. Park, "High-Speed H.264/AVC CABAC Decoding," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 17, No. 4, pp.490 – 494, April 2007.
- [27] 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 *Proc. VLSI-DAT 2007*, pp. 1-4, 2007.
- [28] 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 *Proc. ICASSP 2007*, vol. II, pp.105-108, 2007.
- [29] L. Li, Y. Song, T. Ikenaga, and S. Goto, "A CABAC Encoding Core with Dynamic Pipeline for H.264/AVC Main Profile", In *Proc. APCCAS 2006*, pp 761-764, Dec.2006.
- [30] X.H. Tian, T.M. Le, B.L. Ho, and Y. Lian, "A CABAC Encoder Design of H.264/AVC with RDO Support," In *Proc. 18th IEEE Int. Symp. on Rapid System Prototyping*, pp. 167-173, 2007.
- [31] R.R. Osorio and J.D. Bruguera, "High-throughput architecture for H.264/AVC CABAC compression system", *IEEE Trans. Circuits and Systems for Video Technology*, vol. 16, no. 11, pp. 1376-1384, Nov. 2006.