

A CABAC Encoder Design of H.264/AVC with RDO Support

X.H. Tian, Thinh M. Le, B.L. Ho, Y. Lian
Department of Electrical and Computer Engineering
National University of Singapore
elelmt, g0402624@nus.edu.sg

Abstract

In this paper, a HW CABAC encoder architecture is proposed targeting H.264/AVC main profile. CABAC and Rate Distortion Optimization (RDO) are two important coding tools that enhance coding efficiency of H.264/AVC. However, coding speed of CABAC encoder is limited by the coding data dependence and its serial coding procedure, and RDO requires large memory resource and costs long delay of memory access to backup and restore CABAC context state data. The CABAC encoder of this paper utilizes pipeline structure at top level to reduce data dependence, and coding speed of one symbol per cycle is achieved. A context managing mechanism is designed that fully supports RDO coding of H.264/AVC encoder. It significantly reduces context memory cost and operation delay for context backup and restore. 85% of context memory resource is reduced comparing to the reference design. Synthesis result shows that the encoder can work at 620 MHz targeting 0.13 μ m CMOS process.

1. Introduction

The emerging ITU-T/ISO/IEC standard H.264/AVC [1], [2] for video compression aims at enhanced coding efficiency and network friendliness over a wide range of transmission systems. Since the establishment of H.264/AVC standard in 2003, research groups from both industry and academic institutes have devoted efforts on system level implementation of codec complying with the standard. A number of new coding techniques adopted in the H.264/AVC standard have significantly improved the encoding efficiency compared to the older standards. The Context-based Adaptive Binary Arithmetic Coding (CABAC) [3] and Rate-Distortion Optimization (RDO) [4] are two of the techniques that significantly enhance coding efficiency of the new standard. CABAC is adopted as the entropy coding tool for main profile and higher profiles of H.264/AVC, and it achieves an average of 9%-14% of bit rate saving over the baseline profile entropy coding method

CAVLC of the standard. RDO also achieves an average of over 13% bit rate saving in CIF and SDTV tests, and the combination of CABAC and RDO reduces around 20% of encoding bit rate [5]. Despite the benefit from CABAC and RDO, computation complexity of video encoder is significantly increased and one third of total dynamic instructions are used by the combination of RDO and CABAC operations even when fast RDO is used [5]. Therefore, it is desirable to design HW IP block to accelerate CABAC and related RDO coding in both SoC-based and all-HW H.264/AVC encoder in high quality video coding application.

Several CABAC encoder designs have been proposed recently. Reported by *Ha et al.* in [6], parallel processing and pre-fetch technique are adopted to achieve coding speed of 5 clock cycles per bin, at clock frequency of 18~54MHz. The processing speed is low for high definition video coding. *Shojania et al.* [7] separates renormalization and bit output stages of CABAC to reduce coding delay, and achieve renormalization in one cycle. However, because context memory access takes long procedure, the encoder only achieves coding speed of 3 cycles per bin. Reported by *Li et al.* in [8], dynamic pipeline scheme is used to improve throughput of CABAC encoder. But the data dependence of arithmetic coding and renormalization is not removed, so the pipeline cannot fully work, and it takes an average of 1.7 cycles to encode one bin. *Osorio et al.* [9] uses 4x4-byte small cache to pre-fetch context models in order to reduce memory access time. For the case of cache hit, the encoder achieves speed of one cycle per bin. But when there is a cache miss, more than one cycle is needed. In the designs of [6], [7], [8], and [9], RDO is not supported, thus, these designs cannot be applied when RDO is required to enhance video coding quality. Reported by *Osorio et al.* in [10], Range and Low are separately updated in arithmetic coding, and process units are doubled. Coding speed is around two bins per cycle on the cost of lower frequency and more complicated control logic. RDO encoding is considered in the design, with one more memory block for RDO context updating. However, the scheme cannot support P8x8 sub-block RDO mode decision, which

is significant to inter-prediction performance of H.264. In [5], three large FIFO buffers are used to backup context model states during RDO mode decision procedure. However, the timing delay to restore RDO state from backup FIFOs is not small, and area occupied by the FIFOs is significant to the ASIC design.

In this paper, a SoC-oriented [11] top-level fully pipelined CABAC encoder targeting main profile of H.264/AVC standard is designed to accelerate coding speed. Various RDO related operations are implemented in the encoder structure. A scheme of RDO context state backup and store is designed that achieves both timing and area efficiency. In the following sections, CABAC encoding procedure and related RDO operations will be briefly introduced first. Then, each part of CABAC encoder design will be discussed. Experimental implementation results and conclusions are given in the last two sections.

2. Introduction of CABAC encoding and related RDO operations

As introduced in [1] and [2], the encoder of H.264/AVC standard consists of three main components: binarizer, context modeler, and binary arithmetic coder. (1) Binarization maps all non-binary valued syntax elements (SE) into bin sequences called bin strings. Five binarization schemes are used in CABAC: Unary (U), Truncated Unary (TU), k^{th} order Exp-Golomb (EGk), concatenation of the first and third scheme UEGk, and fixed length binarization (FL). (2) The context modeler selects a probability model from a pre-defined set of probability models for each bin of SE bin string based on context index (CtxIdx) of the bin when regular coding mode is used. The fetched context model indicates the most probable symbol (MPS) and probability state index (pStateIdx) of the bin. (3) The binary arithmetic coder performs arithmetic coding of each bin using selected probability model. The principle of arithmetic coding is based on recursive sub-division of interval length which is defined by low bound (Low) and length (Range) of the interval. Two coding modes are used: regular bin coding which uses context models, and bypass bin coding for bins with equal probability of 0 and 1. In regular bin coding, bit 7 and 6 of Range and pStateIdx are used to lookup the Range of LPS (least probable symbol) instead of complex multiplication. To keep the precision of the interval, Range and Low are renormalized when the Range is small, and the higher bits shifted out of Low are packed and output.

Rate Distortion Optimization is adopted by H.264/AVC encoder [12] to select best encoding mode for each macroblock (MB) to achieve lowest rate-distortion cost. CABAC coder is used to calculate the rate of each RDO mode. Because of larger number of possible coding modes, computation complexity of CABAC encoding and RDO mode decision is high. During RDO coding, all context models need

to be backed up before coding a new mode, and restored after the mode coding in most cases. The context backup and restore operation causes huge amount of memory access operations and occupies large percentage of processor computation time in SW implementation. 3 large backup memory blocks are allocated in JM software [12] to store the intermediate context state during P8x8 sub-block mode decision. Therefore, reduction of context backup and restore operation time and backup memory size is significant to CABAC encoder.

3. Design of CABAC encoder

The top level HW structure of this CABAC encoder is shown in Figure 1. The encoder consists of 3 major function partitions: first stage, second stage, and context manager (CM).

The first stage consists of three functional units: (a) SE parameter and ctxIdxInc parsing, (b) input parameter parsing and binarization, and (c) encoding bin string serial output. 4 small FIFOs are also used to buffer the coding parameters between the functional blocks.

Input data packets received from system bus interface are first parsed in unit (a), in which ctxIdxInc (context index increment) packets are sent to the 4x32-bit FIFO (2) and the other input packets are buffered in the 21x8-bit FIFO (1). CtxIdxInc value is buffered separately because it is not used in unit (b), but in unit (c) with output parameter from unit (b). Unit (b) controls CABAC encoding procedure of the encoder because all coding data and control parameters are parsed here and sent to different function blocks. SE data are binarized, packed with bin type and ctxOffset (context offset), and sent to FIFO (3). Unit (c) reads in bin string from FIFO (3) and serially packs each bin in the string with bin type information. If it is regular bin, CtxIdx is also packed which is the sum of ctxOffset and ctxIdxInc from FIFO (2). In comparison, bypass bin and EOS (end of slice) bin are packed with no ctxIdx. RDO operation parameters are also packed in unit (c). Output packets from unit (c) are buffered in 13x32-bit FIFO (4). All the 4 FIFOs support simultaneous read and write operations in order to accelerate data access speed.

The second stage of CABAC encoder takes charge of context access and binary arithmetic coding. It consists of 3 function units: (d) context access (CA), (e) arithmetic coding and renormalization, and (f) bit packing.

Context model of regular bin is accessed and updated in CA, and RangeLPS of regular bin is fetched. The other bins and RDO control parameters are only buffered in CA. Unit (e) encodes regular bin, bypass bin, and EOS bin in different coding procedures. Range and Low are renormalized, and output parsing string is sent to unit (f). Unit (f) packs coded bits and outputs packet, or sends out coding rate of each RDO mode during RDO coding.

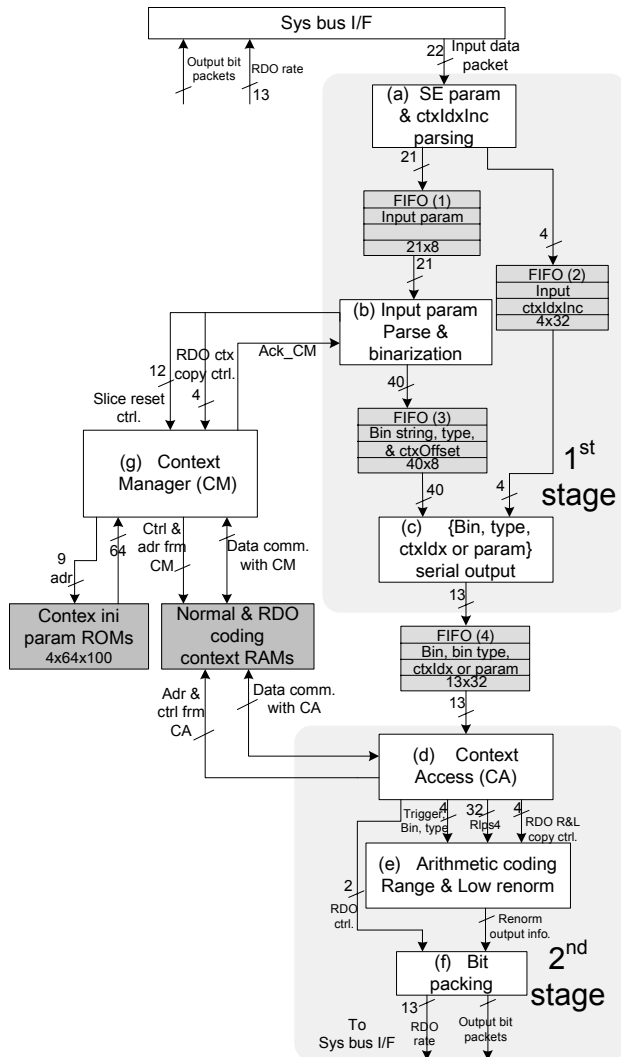


Figure 1: Top level structure of CABAC encoder

Unit (d), (e), and (f) of second stage consists of 3-stage pipeline. Each unit processes one input data and output one coded parameter every cycle. If FIFO (4) is not empty, the second stage ensures constant coding speed of one bin per cycle. In the first stage, unit (a) and (c) can all output one packet every cycle if respective output FIFO buffers are not full and input packet is available. Unit (b) can output one packet per cycle for non-residual SEs. For residual SEs, unit (b) needs to pause to collect all residual input data of the coding block. However, if FIFO (3) and FIFO (4) are not empty before unit (b) output the first residual packet, the coding pipeline of second stage is not influenced.

The third part of CABAC encoder is unit (9) context manager (CM). CM initializes context models of each new slice, backs up and restores context models during RDO coding. CM receives instructions from unit (b), reads pa-

rameters from context initialization ROMs, and access the normal or RDO coding context RAMs.

3.1. Input parameter parsing and binarization

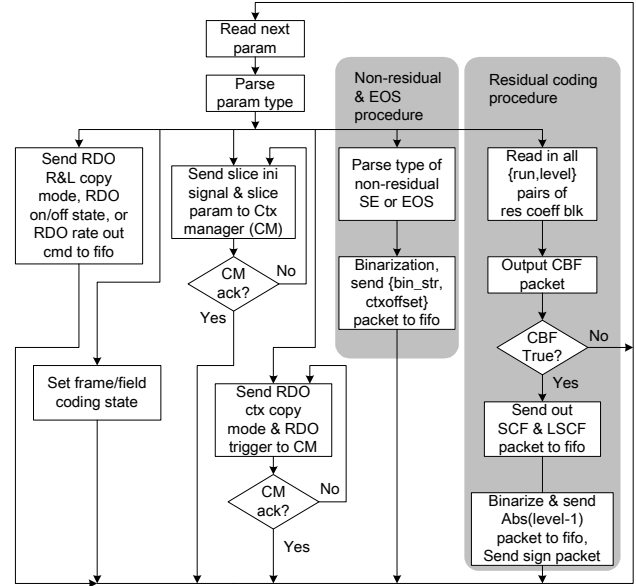


Figure 2: Unit (b) processing flow of input parameter parsing and binarization

As shown in Figure 2, unit (b) parses each input packet to several types through packet head. Context initialization instruction is sent to CM with slice head information. Instruction of RDO context copy is also sent to CM. Before CM finishing the operation, unit (b) waits for the ack signal. State flag of frame/field coding is parsed and stored in unit (b), which is used to select ctxOffset during residual SE processing. RDO operation parameters including Range and Low backup and restore mode, RDO on and off state, and RDO rate output instruction are packed with bin type indicator, and sent to FIFO (3).

Non-residual SE and EOS flag are processed in one procedure. Different SE types or EOS flag are identified by the 4-bit sub-type value in input packet. One of the five binarization methods is chosen to binarize each SE parameter. Because ctxOffset of bin string is also packed in the same output packet, in order to simplify packet parsing and processing procedure of unit (c) (bin serial output), prefix and suffix bin strings are separately packed if the SE has suffix bin string. For intra-coded MB type, because the 2nd bin is coded as EOS bin, the bin string needs to be partitioned and sent to unit (c) separately if it has more than 1 bin. Suffix string binarization of motion vector difference is implemented by fast Exp-Golomb coding circuit, which supports 11 bit suffix input range.

For residual SE processing, input {run, level} pairs of one residual coefficient block are read in first, and SE values of CBF, SCF, LSCF, absolute coefficient level, and sign are generated during read procedure. After reading, the packets of residual SEs are sent out step by step. If CBF is 0, no residual data need coding, and no more residual packets are sent out. Because the theoretical max range of absolute level is 13 bits, fast Exp-Golomb coding circuit is design and it supports max output bin string length of 25 bins.

Each bin string in the packet occupies 25 bits (max range) with additional 5-bit index of MSB to notify unit (c) the position of first bin to process in the string.

3.2. Context Access Unit

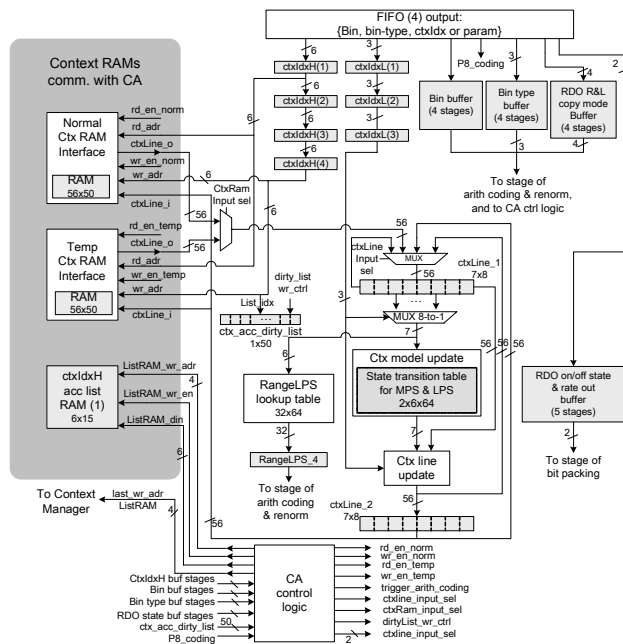


Figure 3: 4-stage context access pipeline structure

Context Access unit (CA) is the first coding step of 2nd stage pipelined encoding structure. The HW structure of CA is shown in Figure 3. 13-bit input packet of {bin, bin type, ctxIdx or control parameter} are parsed according to bin type value. For regular bin, as shown in figure, the 6-bit higher part ctxIdxH of the 9-bit ctxIdx is buffered in 4 pipeline stages and 3-bit lower part ctxIdxL is buffered in 3 stages. The 1st stage ctxIdxH(1) addresses one line of context RAM containing 8 context models (56 bits) that needs to fetch. After 2 cycles, context line is read out from RAM and is ready for use. ctxIdxL(3) selects one context model from the 56-bit context line buffer ctxLine₁. The 6-bit pStateIdx of the selected context model is used to lookup the 4 possible values of Range LPS which are stored in 32-

bit RangeLPS₄ and referenced by arithmetic coder of unit (e). The context model is also updated through one of two 6x64-bit state transition tables according to whether the bin is MPS or LPS. The updated context model is concatenated with the other 7 unchanged models and stored in 56-bit buffer ctxLine₂. Input bin, bin type, and copy mode of RDO Range and Low are buffered in 4-stage buffers and sent to the unit (e). RDO on/off state and rate output instruction are buffered in 5 stages, and sent to unit (f) for bit packing.

All 399 context models are stored in 50 lines of normal context RAM, 8 models each line. The advantage of using two context line buffers ctxLine₁ and ctxLine₂ is that 16 context models are buffered in CA. If the next request context model is located in these two lines, RAM read operation is not taken. Only when the addresses of the two lines are different, CA will write ctxLine₂ to context RAM. Because in many situations, context models which have been consecutively accessed are located in the same or neighboring line, RAM read and write frequencies can be significantly reduced, and dynamic power of RAM access is also reduced. During RDO coding, another temporary RAM of same size of normal RAM is used to store updated context lines. During coding of one RDO mode, if the line requested is not updated, it is read from normal RAM; otherwise, it is read from temp RAM. And updated line is stored to the temp RAM. A 50-bit ctx_acc_dirty_list records which line has been updated. During non-P8x8 RDO mode coding, the normal context RAM is unchanged, so the time of context restore operation is saved. For P8x8 sub-block mode decision of RDO coding, the address of context line been modified is stored in 6x15-bit RAM(1), a ctxIdxH access list according to ctx_acc_dirty_list. This RAM is further used by context manager in RDO context copy operation.

CA control logic controls the read and write enable signals of two context RAMs, selection of read in context line, update of dirty list, and selection of update sources of ctxLine₁. The 4-stage CA pipeline takes 4 cycles to read in, update and write back a context model. However, the pipeline processing speed is one context model per cycle.

3.3. Arithmetic Coding, Renormalization, and Bit Packing

Unit (e) arithmetic coding and renormalization and unit (f) bit packing in Figure 1 are the 2nd and 3rd stages of the 3-stage encoding pipeline. The structure of these two stages is shown in Figure 4. In unit (e), regular bin, bypass bin, and EOS flag are processed in 3 different procedures.

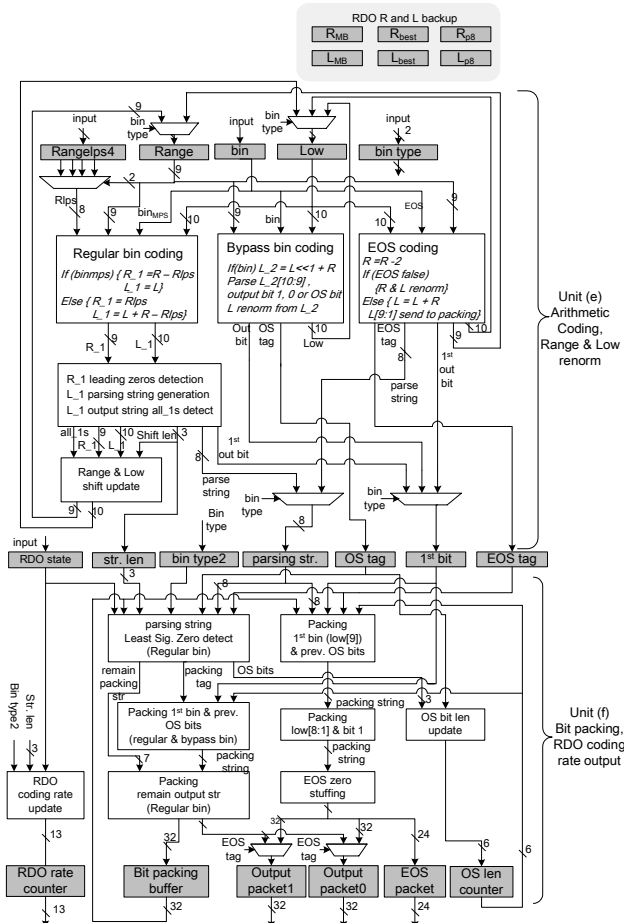


Figure 4: Pipeline structure of arithmetic coding, renormalization, and bit packing

In regular bin coding, one of the 4 input RangeLPS values is selected based on Range[7:6]. Coding interval is updated to one of its two sub-intervals based on whether the bin is MPS or LPS. Range and Low of the interval are updated accordingly. If Range is smaller than 256, Range and Low need to be renormalized. An efficient renormalization and bit output scheme is proposed in this paper, which is different from original time-costing sequential shift method defined in the standard. As shown in Figure 5, N bits of leading zeros of Range are detected first. The N+1 top bits of Low are sent to unit (f) including the first output bit and remaining N bits of parsing string. Then both Range and Low are left shift N bits. If top N bits of Low are not all 1s before shift, Low[9] is set to 0. The renormalization of Range and Low is then completed.

In bypass bin coding, Range is not changed, and Low can output one bit of 0 or 1, or one outstanding (OS) bit. OS tag indicates the output is OS bit, while output bit is sent to register of 1st output bit. In EOS flag coding, Range is first subtracted by 2, then two sub-branches may be taken.

If EOS is true, Low is added by the new Range and top 9 bits are output for packing and EOS tag is set true; otherwise, regular bin renormalization procedure is taken.

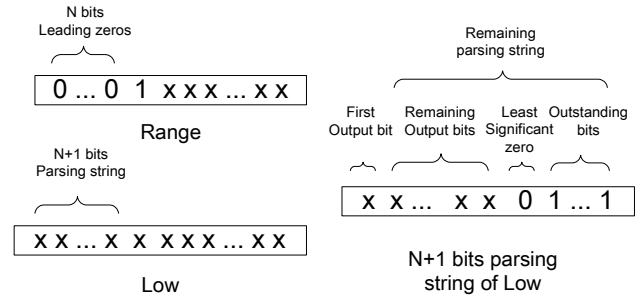


Figure 5: Scheme of Range and Low renormalization and bit output for regular coding bin

During RDO coding, Range and Low also need to be backed up or restored when RDO mode switches. According to the parameter of RDO copy mode of Range and Low from CA, unit (e) copies data between coding Range, Low, and 3 backup pairs of Range and Low in 7 modes, including 5 modes used only in P8x8 RDO coding.

Unit (f) receives RDO state value from CA. If RDO is on, coding rate of each RDO mode is accumulated according to the length of input parsing string and bin type from unit (e). When RDO rate output instruction is received from CA, a 13-bit RDO coding rate is sent out to system bus interface and the rate counter is cleared.

When RDO state is off, the encoder output bit stream in two paths: EOS packing or non-EOS packing. (1) For non-EOS packing, the remaining parsing string of regular bin coding from unit (e) is parsed to 2 sections: remaining output bit string and new outstanding string (Figure 5) based on detection of least significant zero in the string. Then the 1st output bit from unit (e) and the existing OS bit string are packed to the output string buffer. Maximum length of 63 OS bits is support in the design, which is enough in real application. The remaining output string is then packed. One or two 32-bit packets may be output if the packing string length is 32 or more. The OS bit counter is updated by the new OS string length in the parsing string. If there is no output bit, OS counter is accumulated by new OS string length. (2) For packing of EOS string, output bit Low[9] is first packed with existing OS bits, then the remaining 8 bits Low[8:1] and one additional bit 1 are packed. If the packing string length is not of integer bytes, additional bit 0s are stuffed into the string. All the bytes in the packing string are output. If the left bytes can not full a 32-bit output packet, they are sent out in 24-bit EOS packet, with 2 bits indicating number of bytes in the packet.

The size of output packet is defined as 32 bits instead of one byte used in JM SW in order to reduce system bus ac-

cess frequency. And it is easier to deal with long OS bits output situation.

3.4. Context Manager

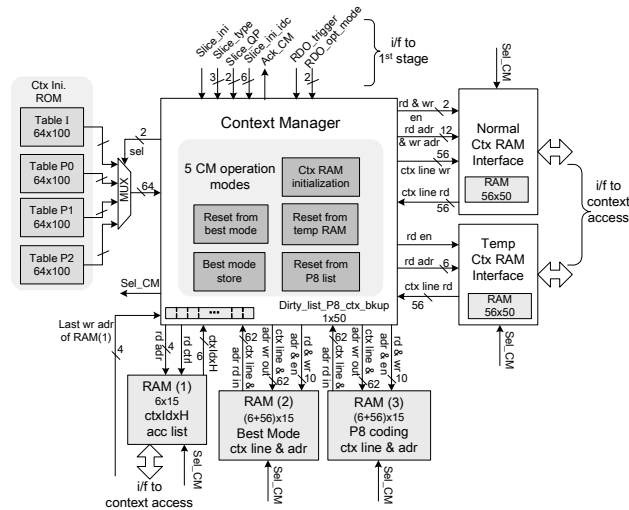


Figure 6: Context Manager with context coding RAMs and context initialization ROMs

Context manager implements two functions: context model initialization for new slice and context model backup and restore operation in RDO coding.

3.4.1. Context model initialization. Comparing to the other CABAC designs, context model initialization is implemented in HW only in this design. The advantage of HW initialization is that HW calculation is time saving and cost of processor calculation and system bus occupation is removed. As shown in Figure 6, one of 4 ROMs is selected to provide context initialization parameters to CM. Each ROM is 64x100 bits, 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: ROM line read address is output; stage 2: the ROM stores read address; stage 3: 4 parameters are read out; stage 4: 4 multiplications are executed; stage 5: 4 context models are generated. 4 processing units are paralleled allocated in this pipeline, so calculation speed is further accelerated. Every two cycles, 8 generated context models are written to normal context RAM. The total 399 context models can be initialized in 104 cycles, including pipeline preparing time.

3.4.2. Efficient RDO context copy operations. As discussed above, because two context RAMs are used to store original and updated context line respectively, no context backup or restore is needed in non-P8x8 RDO coding. However, during P8x8 sub-block RDO mode decision, con-

text coherence of adjacent 8x8 blocks makes it necessary to backup and restore context state data.

By investigating the context models accessed during P8x8 sub-mode decision, it is clear that less than 15 context lines can be modified during the whole procedure of P8x8 sub-block mode decision. Accordingly, a 50-bit dirty list and three small RAMs (Figure 6) are allocated:

- RAM(1): 6x15-bit ctxIdxH_acc_list stores all line addresses of context lines modified by CA during each mode of 8x8 sub-block coding;
- RAM(2): (56+5) x15-bit best list stores modified context line and line address of best mode of current 8x8 sub-block;
- RAM(3): (56+5) x15-bit P8 list backs up original values of all context lines modified in the normal context RAM and corresponding line addresses.

4 types of necessary context copy procedures are designed in CM (Figure 6): (a) Best mode store: read line address from RAM(1), and read context line from temp RAM, then store both address and context line to RAM(2); (b) Reset from best mode: update context state of normal RAM from the context line stored in RAM(2), and do necessary backup of context line and address to RAM(3) according to the dirty list; (c) Reset from temp RAM: after the last mode of sub-block coding, the best mode context can be stored in temp RAM, if true, read line address from RAM(1), and fetch modified best mode context line from temp RAM and write to normal RAM, but do necessary backup of address and context line of normal RAM to RAM(3) before modification according to the dirty list; (d) Reset from P8 list : after all P8x8 sub-block-mode coding finishes, the normal RAM is restored to the original state before P8x8 RDO coding by recovering from the address and context line stored in RAM(3).

All these 4 types of context copy procedures are designed by pipeline structure to process one context line per cycle. Because in each cycle, one context line of 8 models are simultaneously backed up or restored, and the number of lines been modified can be significantly smaller than 15, the context model copy operations of P8x8 RDO sub-block mode decision is faster than that of [5], and time cost of all non-P8x8 RDO context copy operations is saved.

4. Circuit Simulation and Synthesis

The CABAC encoder is designed at RTL level using Verilog HDL. It is simulated and verified by testing standard CIF and QCIF sequence parameters generated by JM 9.3 software [12]. The CABAC encoder is synthesized targeting 0.13um CMOS process. Synthesis results show that the encoder can work properly at clock frequency of 620MHz at normal condition (1.2v, 25C), corresponding to the coding speed of 620Mbin/sec. The synthesized circuit

area of each component of the encoder is listed in Table 1. Area is measured by equivalent 2-input NAND gate count.

Table 1: Synthesized area of CABAC encoder

Unit name	Area in Gates	
Unit (a) input classification	176	1 st stage 4.9K
Unit (b) parsing & binary	4,289	
Unit (c) bin serial out	442	
Unit (d) CA	3,363	2 nd stage 14.5K
Unit (e) arith. & renorm.	1,704	
Unit (f) bit packing	9,472	
Unit (g) CM	6,199	
Memory interface	1,810	
Total area of encoder	27.5K	

Table 1 shows that the area of CABAC encoder is 27.5K gates. The 1st stage of input parameter parsing and binarization occupies 4.9K gates, or 17.9% of total area. The 2nd stage of CA and binary arithmetic coder occupies 14.0K, or 51.46% of total area. Context manager occupies 6.8K (25.01%) of encoder area, and memory interface of RAM, ROM and FIFO buffer costs additional 1.8K (6.6%).

The total size of 5 context RAMs is 7.6K bits. In comparison, design [5], which also supports RDO coding, utilizes 52Kbits of memory resources to store the coding and backup context state data. Therefore, 85% of context memory is reduced in our design. In this paper, additional 1Kbits of FIFO buffers are used in the 1st stage, and 25.6Kbits of ROM stores context initialization tables. As parameter parsing, binarization and context initialization are not implemented in most of other designs, the memory cost of FIFO and ROM cannot be compared.

5. Conclusions

In this paper, a CABAC encoder at main profile of H.264/AVC standard is designed. The encoder is designed with maximum pipelining, and coding speed of one cycle per bin is achieved, which is better than the coding speed of 5, 3, and 1.7 cycles/bin reported in [6], [7], and [8]. Design [5] achieves coding speed of 1 bin/cycle and clock frequency of 520M to 650MHz targeting 0.13um process, which is similar to the coding speed in this design.

RDO related CABAC operations is not supported in design [6], [7], [8] and partially supported in [10]. In comparison, RDO related CABAC operations are fully supported in this encoder including fast and memory efficient RDO context backup and restore operations. Comparing to design [5], 85% of context memory resource is reduced in this design. Parallel and pipelined HW context initialization is also proposed in this design. The encoder is verified and synthesized using 0.13um CMOS process. Synthesis results show that clock frequency of 620 MHz is achieved, and the

encoder occupies 27.5K logic gates, 8.6K RAM bits, and 25.6K ROM bits.

References

- [1] ITU-T Recommendation H.264, *Advanced video coding for generic audiovisual services*, May, 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] 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.
- [4] G. J. Sullivan, T. Wiegand, "Rate distortion optimization for video compression", *IEEE Signal Processing Magazine*, pp. 74-90, Nov. 1998.
- [5] J. L. Nunez, *et al.*, "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, May 2006.
- [6] V.H.S. Ha, W.S. Shim, and J.W. Kim, "Real-time MPEG-4 AVC/H.264 CABAC Entropy Coder," in *Digest of Technical Papers*, International Conference on Consumer Electronics, pp.255 – 256, 2005.
- [7] H. Shojania, S. Sudharsanan, "A High Performance CABAC Encoder," *Proceedings of the 3rd International IEEE-NEWCAS Conference*, pp.315 – 318, 2005.
- [8] L. Li, Y. Song, T. Ikenaga, and S. Goto, "A CABAC Encoding Core with Dynamic Pipeline for H.264/AVC Main Profile", *ASPCAC2006*, pp 761-764, Dec.2006.
- [9] R.R. Osorio and J.D. Bruguera, "Arithmetic Coding Architecture for H.264/AVC CABAC Compression System," *Euro-micro Symposium on Digital System Design*, pp.62 – 69, 2004.
- [10] R.R. Osorio and J.D. Bruguera, "High-Throughput Architecture for H.264/AVC CABAC Compression System", *IEEE Transaction on Circuits and Systems for Video Technology*, Vol. 16, No. 11, pp.1376 – 1384, Nov. 2006.
- [11] T.M. Le, X.H. Tian, B.L. Ho, J. Nankoo, and Y. Lian, "System-on-Chip Design Methodology for a Statistical Coder," *Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping*, pp.82-88, 2006.
- [12] Reference codec software: JM 9.3. Available at: <http://iphome.hhi.de/suehring/tml/>