A Five-stage Pipeline Design of Binary Arithmetic Encoder in H.264/AVC

Song Rui and Cui Hongfei and Li Yunsong and Xiao Song
Xidian University, Xi'an, 710071,Peoples R China
E-mail: rsong@xidian.edu.cn Tel/Fax: +86-29-88200187

***Abstract*—Context-based Adaptive Binary Arithmetic Coding (CABAC) is a well known bottleneck in H.264/AVC encoder. Despite its high performance, the tight feedback loops make it difficult to parallelize. Most researchers are concerned about multi-bin processing regardless of the pipeline design. But without pipeline, the overall performance is greatly limited. In this paper, the critical path for hardware implementation of binary arithmetic encoder (BAE) was analyzed in detail. We break the computing steps to the best extent, and re-arrange it to the appropriate pipeline to get a balanced latency at each stage. Further, new binary arithmetic encoder architecture with five stage pipeline and 1 bin per cycle was proposed, the latency of critical path were cut off exceedingly, and the frequency and throughput rate was improved. An FPGA implementation of the proposed pipelined architecture in our H.264 encoder is capable of 190Mbps encoding rate. And a maximum 483MHz could be achieved on SMIC 0.13μm technology, which meets the requirements of QFHD encoding at 30fps. The proposed architecture could be utilized in other designs to get a better performance.**

I. Introduction

High definition and high speed video are of the most promising applications for future consumer electronics. Frame resolution as high as Quad Full HD becomes much more prevalent in recent years along with the rapid innovation of silicon technology[1]. For hardware implementation, parallel processing is a commonly used technique to improve performance. But in H.264/AVC, the well known bottleneck owing to highly serialized calculating of context adaptive binary arithmetic coding is hard to be parallelized. Complex steps were involved to encode each one bit for CABAC, including binarization, context modeling and binary arithmetic encoding. Besides, a massive probability model should be maintained and updated before the next bit encoding. Strong correlation exists between contiguous bits, therefore, the incoming bit could not be correctly coded until all the necessary computing and updating process for prior bit ended. These properties make it quite challenging to parallelize the encoding process to improve throughput rate. Most researches focused on how to process more than 1 bit in average by the statistical encoding property, but limited attention was paid on optimization of the pipeline architecture to promote frequency performance.

The encoder designed by R. R. Osorio[4] could attain more than 2 bin encoding per cycle on average. To accelerate encoding, efficient binarizer and optimized storage strategy of context model were presented. Nevertheless the control logic is too complex, and only two stage pipeline is used, which degrades the overall performance. The proposed encoder was implemented on an Virtex-II 2000 FPGA and the a maximum frequency of 92MHz could be achieved. In Ref.[5] by F. Wei, with adjustment to the data flow, the operator low is replaced by "range+low", and some redundant calculation was reduced. However, without pipeline architecture, the critical path would hardly be shortened. A RDO supported cabac encoder is given by T. XiaoHua[6], efficient memory access scheme is proposed to reduce context ram access frequency, but traditional three stage pipeline is used in binary arithmetic coding. The latency of unit(e) is considerably longer than each stage of unit(f), hence for binary arithmetic coding, the frequency performance is not optimized. The pipeline architecture was advanced by Z. Wei.[7], in which four-stage pipeline were applied and the maximum frequency could achieve 300MHz on .18μm SMIC standard cell lib. Nonetheless in our opinion, the critical path could be further cut off. A six stage pipeline cabac encoder is proposed in Ref.[8], all the design strategies on pipeline focused on multi-bin processing. However, for hardware design, the benefit of efficient pipeline is neutralized by complex processing of multi-bin, then the overall performance is degraded.

The authors doesn't focus on pipeline design, therefore pipeline is not optimized intentionally. In this paper, we analyzed the critical path of binary arithmetic encoder in detail. Computing steps of binary arithmetic coding were broken and re-arranged to separate pipeline to get a balanced latency. Then, a five-stage pipeline architecture were proposed. The critical path were cut off to a great extent, as a result, the maximum frequency could be up to 190MHz on Virtex-V FPGA. Synthesis report given by synopsys design compiler on SMIC 0.13μm shows a maximum 483MHz frequency, which fulfills the application of QFHD encoding.

II. Pipeline Analysis of CABAC Encoder

The CABAC encoding process in H.264/AVC consists of four parts as Fig. 1. Before encoding one slice, the encoder were firstly initialized, which including initialization of binary arithmetic encoder and probability model for each syntax element(SE). Then the input SE was binarized to get the bin string. Specific probability model was selected and input to binary arithmetic encoder, then gets the output bit stream.

This work is partially supported by the Fundamental Research Funds for the Central Universities; the 111 project under Grant No.B08038; Program for Changjiang Scholars and Innovative Research Team in University.

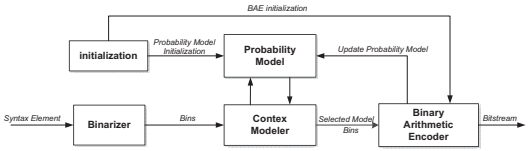

Fig. 1. Block diagram of cabac encoder in H.264/avc

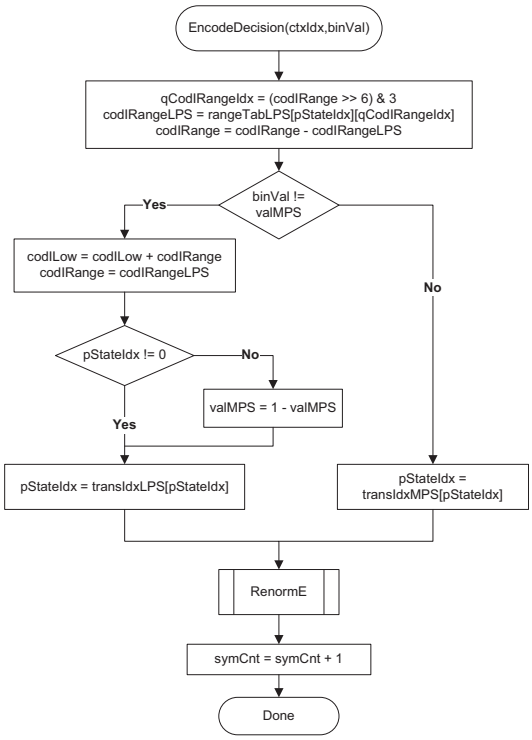

Fig. 2. Flowchart of binary arithmetic encoding process

The probability model should be immediately updated after encoding. For CABAC encoder, SE of each MB could be feed in parallel, as well as the binarization process. But at binary arithmetic encoding stage, all the bin string should be input and encoded sequentially. Thus, the binary arithmetic encoder is the most critical block, and affects the overall performance. BAE consists of probability model fetch, update of range and low interval, bit generation, and probability model update and store, as Fig.2 shows. Limited by the normative syntax, the encoding process of the rear bit in binarized continuous bins should not be started before the end of prior bit. This necessity is the hindrance of parallel computing. But actually, not all the computing results of prior bits are dependent. The complex computing process could be spited into several tiny steps. For rear bit on step n, if only the result of step n of the front bit and the $(n-1)$th step result of current bit is dependent, one pipeline could be inserted. If all the pipeline stages are balanced, the overall performance could be improved. As Fig.2 shows, the update of range and low is independent of context model update, and could be parallelized. For hardware implementation, these two processes could be placed into one

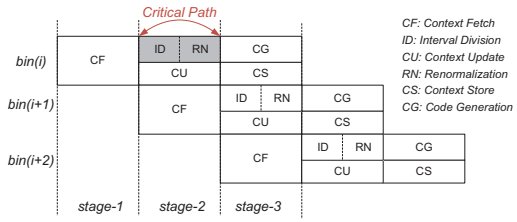

Fig. 3. Directly mapped three stage pipeline

pipeline stage, and computed simultaneously. The pipeline could be roughly mapped as Fig.3, in this three stage pipeline, the context model of the current bin were fetched at the first stage. Then, in the second stage, the arithmetic coding state and context model were updated. The third stage accounts for bit code generation and context store. For the pipeline in Fig.3, most calculation is completed in the second stage.It increases the latency greatly, and constitutes the critical path. To attain better throughput performance, the pipeline should be improved.

III. Binary Arithmetic Encoder With Five-stage Pipeline

In this section, three ways were presented to improve the encoding performance of CABAC encoder. By preprocessing of range update, split and re-arrange of range and low and simplification of the normalization process, the pipeline was optimized at each stage. Critical path was cut off progressively, and the maximum frequency was finally advanced.

A. Preprocess of Range Update

From the flowchart in Fig.2, to update the “range”, we have to look up the two dimensional rLPS table firstly. The dimensional index is probability state index “pStateIdx” and range index “RangeIdx”. In the pipeline architecture of Fig.3, pStateIdx is ready after context fetch process in the first stage. But the RangeIdx could not be obtained until the end of range update process. Extra time was wasted at the current stage when pStateIdx is waiting for RangeIdx to look up the table. Hence, the looking up process of rLPS table could be divided into two steps, at the first step, one of the four rLPS tables indexed by the first dimensional index pStateIdx was selected. And at the second step, the certain rLPS was found. Then the two steps could be divided by one pipeline stage, which will decrease the latency. Meanwhile, the decision on whether the current bin is MPS could also be broken away from the second stage to further reduce path delay.

In order to separate the two interested step from the second pipeline, a new pipeline has to be inserted between the first and second stage. The new inserted pipeline stage should not affect the context update which placed in the former second stage, so the context update was kept at the second stage in new pipeline to reduce the access confliction on probability model. Revised pipeline was depicted in Fig.4, the preprocess for interval division along with context update were placed at

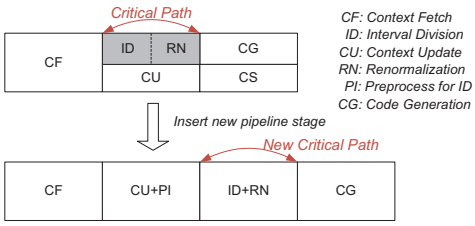

Fig. 4. Four stage pipeline with preprocess for interval division stage inserted

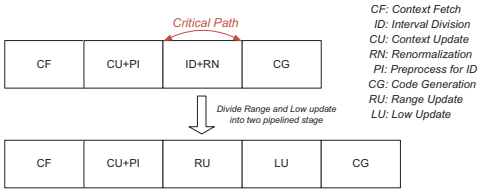

Fig. 5. Five stage pipeline with range update and low update split into two stages

the second stage, and the remainder process of interval division accompanied by renormalization process was accomplished at the third stage.

B. Split and re-arrange of Range and Low Update

At the third stage in the revised four stage pipeline, when the input bin is MPS, no action was needed for “low”, when the bin is LPS, the renewed “low” should be “low+rMPS”, in which rMPS is “Range-rLPS”. Since the low update process is based on rMPS, which is an intermediate result of the updated Range. And for the next bin, range update has no connection with low update of the prior bin, thus, to decrease the update delay of low, the range and low update could also be split into two steps and re-arranged at two pipeline stages. As Fig.5 shows, the third stage was further divided into two new pipeline stages. At the refreshed architecture, range update is placed at the third stage, and the low update at the fourth stage. At the renewed fourth stage, the calculation result of rMPS at range update slot could be directly reused, and only some addition process were needed at low update process. As a consequence, the critical path in Fig.4 was further cut off.

C. Renormalization Using LUT

After each update of Range and Low, the renormalization process both involves non-determined number of iterations, as Fig.6 shows. The Iteration has to be finished within one clock cycle, which is not suitable for hardware implementation. Usually the calculation was done by firstly determine the iteration time. Then shift the “Range” and “Low” by specific number of bits. And the key point is to determine the iteration number. One frequently used “leading zero detector” is counting the number of zeros before the first ’1’ of Range.

The problem could be solved by utilizing of look up table. The case could be separated by its corresponding probability state of the input bin. When the bin is LPS, the Range

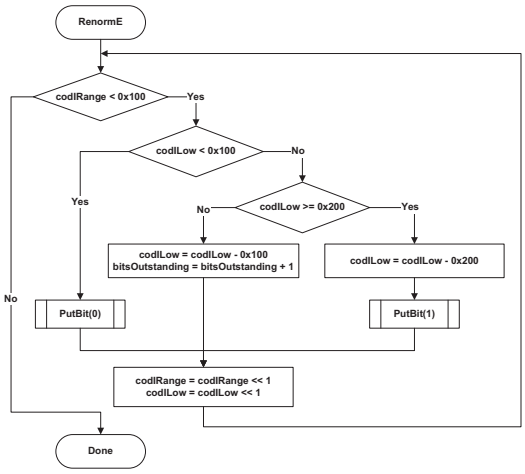

Fig. 6. Flowchart of normalization process

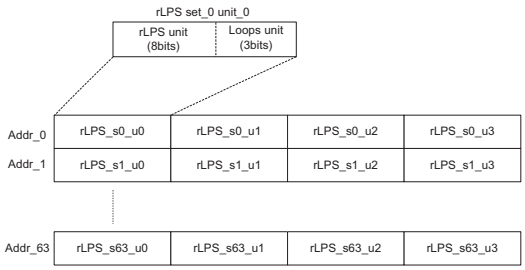

Fig. 7. Expanded rLPS look up table

equivalent to rLPS, then the iteration number could be easily stored accompanied with rLPS table, as Fig.7 shows. We add 3 extra bits to each entry of rLPS table, and the width is expanded to 11 bits. When we are seeking the rLPS in ram, its corresponding iteration number is acquired simultaneously, then left shift Range and Low with this iteration number of bits. When the bin is MPS, the maximum iteration number is 1, so the necessity of renormalization could be determined by the highest bit of Range. If the bit is 1, normalization process was completed by left shift Range and Low with one bit. Otherwise, no normalization is needed.

D. CABAC Encoder with Five-stage Pipeline

Applying all the three techniques above, the five stage pipeline architecture is presented at Fig.8. At the first stage, context was pre-fetched, then the context was updated on observation of input bin at the second stage. The first dimension of rLPS is concurrently located at stage two. After that, Range was updated in advance at the third stage and Low was updated at the next fourth stage. Finally, at stage five, code was generated. In the proposed pipeline architecture, the calculation process is only dependent on the results of the previous stage for current bit and result at the same stage of the preceding bit, therefore, the data dependency constraints were fully met. When the process of one bit were arranged into different pipeline stage, the latency for each pipeline was

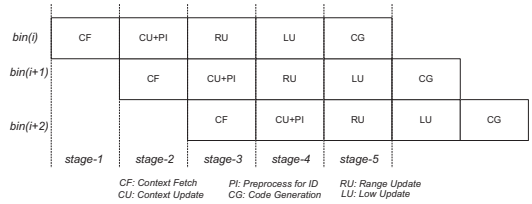

Fig. 8. Proposed five stage pipeline cabac encoder

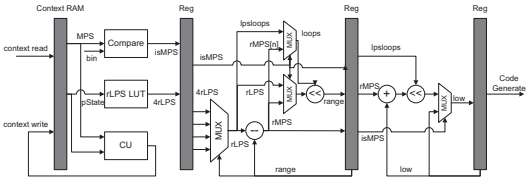

Fig. 9. Hardware architecture of proposed five stage pipeline

balanced extremely. Although the latency for each stage could not be almost equal, for example, the latency of Range update is apparently longer than other stages, nothing could be done further under the syntax constraints in our opinion.

Hardware implementation of the proposed five stage pipeline is designed as Fig.3, context model was stored in context ram, and was read out at the first stage. Input bit was compared with the current MPS, and an isMPS flag is asserted, while one of the four rLPS tables is determined by pState at the second stage. All the information is used to update the context model in the current stage to facilitate the next input bit. At stage three and four, Range and Low are updated separately. Then, fifth stage generates the final bits. The latency is evident in Fig.9.

IV. Implementation

The proposed cabac encoder was implemented on a Xilinx Virtex-V FPGA in verilog HDL, and utilized in our H.264 encoder chip design. The design is further synthesized using Synopsis Design Compiler with .13μm SMIC cell library. The critical path delay on range update in the worst case is 2.07ns, which accounts for a maximum frequency of 483MHz. The architecture is designed to process one bin per cycle, so the equivalent encoding throughput rate is 483Mbps. The comparison to other stat-of-the-art CABAC encoder designs is listed in Table.I. Although the architecture given by F. Wei[5] claimed to achieve 1056Mbps throughput, it is just a equivalent conversion and only part of the design is stated. Besides, the proposed design could get a competitive performance. In addition, the proposed pipeline architecture could also be integrated into other multi-bin designs to enhance their overall performance.

Simulation results on QFHD video sequences were given in Table.II. The test frame rate is 30 fps, gop is set as IBBP, the target output bit rate is set to no more than 240Mbps, which is the maximum allowed bit rate in H.264/AVC main profile level 5.1[2]. The average processing cycle for one macroblock

TABLE I
Comparison of maximum throughput rate

Architecture	Technology	Bin/Cycle	Max. Speed	Max. Th.
R. R. Osorio[4]	AMS 0.35μm	2.3	186 MHz	427.8Mbps
F. Wei[5]	SMIC 90nm	4	264 MHz	1056Mbps
T. XiaoHua[6]	TSMC 0.13μm	1	328 MHz	328Mbps
Z. Wei[7]	SMIC 0.18μm	1	300 MHz	300Mbps
C. JianWen[8]	0.13 CMOSμm	1.42	222 MHz	315Mbps
Proposed.	SMIC 0.13μm	1	483MHz	483Mbps

TABLE II
Clock Cycles Needed Per Macroblock

Video Seq.	Frame Res.	QP	Actual bitrate	Cyc/MB
Bluesky	3840x2176@30	12	235.9 Mbps	469.1
Sunflower	3840x2176@30	12	210.4 Mbps	445.6
Rushhour	3840x2176@30	12	225.2 Mbps	490.3

is not more than 500, hence the proposed architecture meets the requirement of QFHD encoding.

V. Conclusions

A CABAC encoder with five-stage pipeline and 1bin per cycle was presented in this paper, the processing steps are mapped to an custom three step pipeline originally, then critical path along with computing units are broken progressively, the split computing units are finally re-mapped to a five stage pipeline architecture. In the proposed architecture, data dependence between front and rear bits is preserved, and the generated code fully conforms to the H.264/AVC standard. Simulation results show that an average processing cycle number of no more than 500 was needed. The design is implemented on FPGA and synthesized with Synopsys Design Compiler, a maximum of 483MHz was achieved which meets the requirement of QFHD encoding.

References

[1] V. Sze and A. P. Chandrakasan, "A highly parallel and scalable CABAC decoder for next generation video coding," *in Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, 2011, pp. 126-128.

[2] Advanced Video Coding for Generic Audiovisual Services, "ITU-T Recommendation H.264 and ISO/IEC 14496-10 AVC," *Joint Video Team*, 2003.

[3] 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, pp. 620-636, Jul 2003.

[4] R. R. Osorio and J. D. Bruguera, "High-Throughput Architecture for H.264/AVC CABAC Compression System," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 16, pp. 1376-1384, 2006.

[5] F. Wei, Z. Dajiang, and S. Goto, "A high throughput CABAC encoder design," *in Signal Processing and its Applications (CSPA), 2011 IEEE 7th International Colloquium on*, 2011, pp. 99-102.

[6] T. XiaoHua, T. M. Le, J. Xi, and L. Yong, "Full RDO-Support Power-Aware CABAC Encoder With Efficient Context Access," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, pp. 1262-1273, 2009.

[7] Z. Wei, L. DongXiao, S. Bing, L. HoangSon, and Z. Ming, "Efficient pipelined CABAC encoding architecture," *Consumer Electronics, IEEE Transactions on*, vol. 54, pp. 681-686, 2008.

[8] C. JianWen, W. LiCian, L. PoSheng, and L. YounLong, "A high-throughput fully hardwired CABAC encoder for QFHD H.264/AVC main profile video," *Consumer Electronics, IEEE Transactions on*, vol. 56, pp. 2529-2536, 2010.