Design of High Performance Arithmetic Encoder for CABAC in H.264/AVC

Yoonsup Kim*, Jeonhak Moon*, Wonjoon Choi*, Seongsoo Lee*
Seok Lee** Dockha Woo**
*School of Electronic Engineering, Soongsil University, Seoul, Korea
**Intelligent System Research Division, Korea Institute of Science and Technology, Seoul, Korea
wwwys@ssu.ac.kr    http://babel.ssu.ac.kr

Abstract: - CABAC is one of the entropy coding technologies applied to a main profile of H.264/AVC. The subject of concern for performance enhancements of CABAC improves irregular renormalization and bit output of arithmetic coder. This paper proposes an efficient arithmetic encoder for CABAC. To obtain CABAC encoder with high throughput, it adopted a pipeline structure for arithmetic encoder. As a result, at every clock cycle, the input symbol is encoded regardless of the iteration of the renormalization process and so the performance has improved. The proposed architecture was modeled VerilogHDL and was verified through simulations.

Key-Words: - H.264/AVC, entropy coding, CABAC, arithmetic coding, video coding

1 Introduction

H.264/AVC is an international video compression standard [1]. It aims at a wide range of applications such as videophone, HDTV broadcasting, digital multimedia broadcasting (DMB) and internet streaming. It adopted many advanced compression techniques such as advanced entropy coding, strong motion isolation and in-loop deblocking filter. Particularly, important technique for the higher coding efficiency is entropy coding such as context-based adaptive binary arithmetic coding (CABAC) and context-based adaptive variable length coding (CAVLC). CABAC can achieve bit-rate savings of 9%-14% compared to CAVLC [2]. However, it brings additional computational complexity.

This paper proposes an efficient arithmetic encoder for CABAC. To obtain CABAC encoder with high throughput, it adopted a pipeline structure for arithmetic encoder. The arithmetic encoder consists of three coding modes: regular, bypass and termination mode. The proposed architecture was modeled VerilogHDL and was verified through simulations.

2 CABAC Encoding Algorithm

The CABAC encoding algorithm consists of three steps: binarization, context modeling and arithmetic encoding. Fig.1 shows the CABAC encoding block diagram. H.264/AVC defines any meaningful information to be encoded by CABAC as a syntax element. In the beginning of CABAC encoding, the binarization translates a non-binary syntax element to a string of bins. Then, the context modeler reads in the bin string and generates context value according to neighboring data of top and left macroblocks. The context value is an index to the context table built at the beginning of processing a new slice. The context table has 399 entries. The arithmetic encoder processes the bin value from the binarization and context value from the context modeler and outputs encoded bits.

Fig. 1 The CABAC encoder block diagram

2.1 Binarization

Alphabet reduction in CABAC is performed by the application of a binarization scheme to each non-binary syntax element resulting in a unique intermediate binary codeword for a given syntax element, called a bin string. The advantages of this approach are both in terms of modeling and implementation.

There are four such basic types: the unary code, the truncated unary code, the k-th order Exp-Golomb code, and the fixed-length code. In addition, there are binarization based on a concatenation these elementary types. As an exception of these structured types, there are five specific, mostly unstructured binary trees that have been manually chosen for the coding of macroblock types and submacroblock types.
2.2 Context modeling

One of the most important properties of arithmetic coding is the possibility to utilize a clean interface between modeling and coding such that in the modeling stage, a model probability distribution is assigned to the given symbols, which then, in the subsequent coding stage, drives the actual coding engine to generate a sequence of bits as a coded representation of the symbols according to the model distribution.

Four basic design types of context models can be distinguished in CABAC. The first type involves a context template with up to two neighboring syntax elements in the past of the current syntax element to encode, where the specific definition of the kind of neighborhood depends on the syntax element. The second type of context models is only defined for the syntax elements of \( mb_{\text{type}} \) and \( sub_{\text{mb}_{\text{type}}} \). Both the third and fourth type of context models is applied to residual data only. The third type does not rely on past coded data, but on the position in the scanning path. For the fourth type, modeling functions are specified that involve the evaluation of the accumulated number of encoded levels with a specific value prior to the current level bin to encode.

2.3 Arithmetic Encoding

The arithmetic encoder module consists of three coding engines: the regular coding engine, the bypass coding engine and the termination coding engine.

The principle of binary arithmetic coding is to recursively partition the probability interval. As it receives each new symbol, the current probability interval will be partitioned into two sub-intervals. The tag will be updated and pointed to the lower bound of new subinterval according to the encoding symbol. Once the tag is definitely located in either top or bottom half of the interval, a bit will be shifted out to bit-stream. If the tag lies in the upper half the interval, a “1” will be produced; otherwise a “0” will be output. In this way, the base of the new interval can be tracked without waiting for all symbols coded, thus an incremental encoding can be obtained.

2.3.1 Regular Mode

The flowchart of CABAC regular arithmetic coding scheme shows in Fig. 2. The regular coding engine performs recursive interval subdivision. It takes the context value, current encoding bin, variable \( \text{Range} \) and variable \( \text{Low} \) as its input. It gets \( MPS, p\text{StateIdx} \) from the context table and gets \( \text{RangeLPS} \) by looking up the \( \text{rangeTabLPS} \) table. The new \( \text{Range} \) and the new \( \text{Low} \) are calculated based on whether the encoding bin value is equivalent to \( MPS \). Additionally, the new \( p\text{StateIdx} \) is updated by looking up the \( \text{transIdxLPS} \) table and the \( \text{transIdxMPS} \) table. Finally, the regular coder renormalizes the \( \text{Range} \) if it is too small and outputs the encoded bits concurrently.

![Fig. 2 Flowchart for encoding a decision](image)

The renormalization after interval subdivision is required whenever the new interval range no longer stays within its legal range \([2^8, 2^9] \). The process of renormalization is iterative. During each time of iteration of this process, one bit at most in the lower bound of the interval that is used to represent the state of the arithmetic encoder is taken off and inserted into the bit-stream. The combination of two factors makes this renormalization...
compute intensively. First, the algorithm does not know how many times the renormalization loop is going to run. Second, as this process is iterative, it can lead to an unavailable parallelism. In hardware implements, the renormalization tends to consume multiple cycles, thus the throughput is significantly decreased. To attack this problem, the renormalization is rewritten so as to be processed in a single step. The flowchart of renormalization shows in Fig.3.

Fig. 4 Flowchart of PutBit

The Fig.4 is flow chart when there is an output bit occurred by renormalization. The data dependence is very high because it is linked with renormalization, and this loop works. When there is an output bit occurred by renormalization with ‘0’ or ‘1’, the value is just output and if there comes to be bitsOutstanding which has been accumulated so far, it outputs the opposite value of the output first bit. This loop is repeated till bitsOutstanding becomes ‘0’, and it returns in a renormalization loop again if this stage is over.

2.3.2 Bypass Mode

The symbol of same appearance frequency is encoded by bypass mode. The bypass coding engine is used to speed up the encoding process in case of Range - RangeLPS ≈ RangeLPS ≈ Range/2. Because bypass mode encode it without using probability, it is not compressed, but efficiency of the operation improves. Fig. 5 shows flowchart of encoding bypass.

2.3.3 Termination Mode

CABAC encodes it to slice unit. One slice has syntax element of many kinds and when syntax element of the last of the slice is encoded, it is encoded by termination mode. Because it means that it is not the end of the slice if a symbol of last syntax element is ‘0’, it is equal, and it is processed with regular mode and because it is the end of the slice if a symbol is ‘1’, it is treated by flush mode. Flush mode outputs encoded last Low and stop bit. Fig. 6 shows flowchart of encoding termination.

3 The Proposed Arithmetic Encoder

The subject of concern for performance enhancements of CABAC improves irregular renormalization and beat output of arithmetic coder. We proposed the arithmetic encoder which improved the existing complicated renormalization and bit output. Though there must be a Low and a Range to encode a next input symbol according to the algorithm to suggest in a standard document, this value can understand that a
normalization process is over. However, the iteration number of the normalization process is not fixed. Therefore, there is the problem that the clock cycles which is necessary to encode a symbol comes to be different, and the output is not regular.

The proposed architecture let bit generator and renormalization module become independent and worked by pipeline architecture. As a result, at every clock cycle, the input symbol is encoded regardless of the iteration of the renormalization process. Because proposed architecture took data dependence between each stage and removed it, and a total engine worked by a pipeline, the processing of data became early. Fig. 7 shows architecture of proposed arithmetic coder.

The arithmetic encoder consists of three coding modes: regular, bypass and termination mode. It was designed to hardware without processor. Furthermore, termination mode generates a stop bit and a stuffing bit in bit generator after it was worked. The bit generator saves data of arithmetic encoding in FIFO and the saved data generate a last output stream by FSM as Fig.8.

![Fig. 7 Architecture of proposed arithmetic coder](image1)

Fig. 7 Architecture of proposed arithmetic coder

![Fig. 8 FSM state diagram of bit generator](image2)

Fig. 8 FSM state diagram of bit generator

**4 Simulation Results**

Our simulation based on the H.264 reference software JM 9.8 [3]. The proposed arithmetic encoder for CABAC was designed in VerilogHDL and simulated using modelsim simulator. Fig. 9 shows comparison of reference software output with HDL output. To verify the proposed arithmetic encoder, we extracted test vector of input and output with reference software. And we confirmed a correct operation in comparison with the extracted output file from HDL. Furthermore, the renormalization processes the input binary symbol for every clock. And output bit was generated by FSM of bit generator.

![Fig. 9 Comparison of reference software output with HDL output](image3)

Fig. 9 Comparison of reference software output with HDL output

**5 Conclusion**

This paper proposes an efficient arithmetic encoder for CABAC. To obtain CABAC encoder with high throughput, it adopted a pipeline structure for arithmetic encoder. Furthermore, the proposed arithmetic encoder supports three kinds of modes, i.e. regular, bypass, and termination mode. It was designed to hardware without processor. The proposed architecture was verified through simulations using reference software.

**Acknowledgements**

This work was supported by "System IC 2010" project of Korea Ministry of Knowledge Economy

**References:**

