

EXHIBIT A-9

A High Performance CABAC Decoding Architecture (“Yu”)

ANTICIPATES / RENDERS OBVIOUS U.S. PATENT NO. 9,036,701

As described in the following claim chart, Asserted Claims 1, 3–4, 6–7, 8, 10–11, 13–14, 15, 17–18, and 20 of U.S. Patent No. 9,036,701 (the “’701 patent”) are invalid in view of A High Performance CABAC Decoding Architecture (“Yu”), which published in IEEE Transactions on Consumer Electronics Volume 51, Issue 4, in November 2005, and is prior art under at least 35 U.S.C. §§ 102 (a) and (b).

To the extent Complainants allege that Yu alone does not disclose any particular limitation of the Asserted Claims of the ’701 patent, either expressly or inherently, it would have been obvious to a person of ordinary skill in the art as of the priority date of ’701 patent to modify Yu and/or to combine the teachings of Yu with: (1) the knowledge of one of ordinary skill in the art to show all the limitations of the claim; (2) the teachings of the prior art references set forth in any of the references identified in Exhibit A-11.

To the extent that these Invalidity Contentions rely on or otherwise embody particular constructions of terms or phrases in the Asserted Claims, Respondents are not proposing any such constructions as proper constructions of those terms or phrases. Various positions put forth in this document are predicated on Complainants’ incorrect and overly broad interpretation of its claims. Those positions are not intended to and do not necessarily reflect Respondents’ interpretation of the true and proper scope of Complainants’ claims, and Respondents reserve the right to adopt claim construction positions that differ from or even conflict with various positions put forth in this document.

ASSERTED CLAIMS	PRIOR ART REFERENCE: Yu
1. [Pre] A method comprising:	Yu teaches and/or discloses a method comprising the following limitation: <i>See, e.g.,</i> Context Adaptive Binary Arithmetic Coding (CABAC) [2] is one of the two alternative entropy coding method specified in H.264. Compared to the other technique Context Adaptive Variable Length Coding (CAVLC), CABAC entails an access frequency increase from 25% to 30% with bit rate reduction up to 16% [3]. Yu at § I.

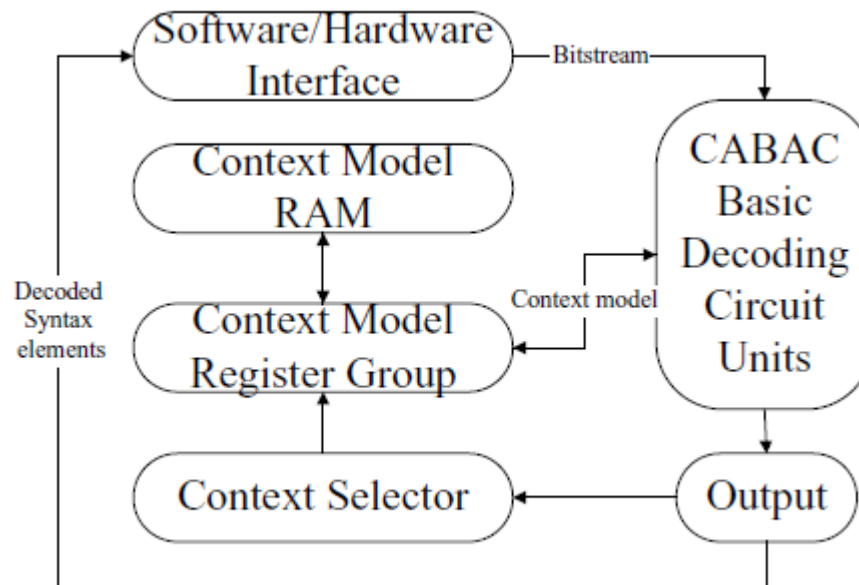


Fig. 1. Framework of CABAC decoder

Yu at Fig. 1.

Actually in CABAC arithmetic encoding implementation, two sub-engines are used: one for bins of regular coding mode as described above, and another so-called bypass coding engine for a fast encoding of bins. For such bins, a default value of 0.5 is used as the estimation of p_{LPS} . The two sorts of bins are referred to as “regular bin” and “bypass bin” in the following text. For more detailed description of CABAC, please refer to [10]-[11].

Yu at § II.

	In this work we have presented a novel architecture for CABAC decoding. CABAC is a key technology adopted in H.264/AVC standard as it offers a 16% bit-rate reduction when compared to baseline entropy coder while increasing access frequency from 25% to 30%. Yu at § V.
[a] determining a frequency of occurrence threshold based on an expected frequency of occurrence of syntax elements in a bit stream;	Yu teaches and/or discloses determining a frequency of occurrence threshold based on an expected frequency of occurrence of syntax elements in a bit stream: <i>See, e.g.,</i> After exploring the binarization modes of different syntax elements and their occurring frequency during the decoding process, we find there are two classes of syntax elements. The syntax elements in the first class contain a few bins and occur only once or a few times in one MB; while the syntax elements in the second class contain several bins and occur tens or hundreds of times in one MB. Syntax elements of the first class contribute only a small part to the arithmetic decoding workload, so in our architecture, one bin is processed per cycle for such kind of syntax elements to decrease the complexity and hardware cost. However to accelerate decoding of the syntax elements in the second class, a new architecture that can decode two regular bins together with one bypass bin in one cycle is proposed. Those syntax elements in the second class include abs_mvd, significant_coeff_flag, last_significant_coeff_flag and coeff_abs_level_minus1. Yu at § III.B.

[b] categorizing a plurality of syntax elements of video content into first and second categories based on the frequency of occurrence threshold, wherein syntax elements which occur greater than the frequency of occurrence threshold are categorized into the second category and syntax elements which occur less than the frequency of occurrence are categorized into the first category;¹	Yu teaches and/or discloses categorizing a plurality of syntax elements of video content into first and second categories based on the frequency of occurrence threshold, wherein syntax elements which occur greater than the frequency of occurrence threshold are categorized into the first category and syntax elements which occur less than the frequency of occurrence are categorized into the second category: <i>See, e.g.,</i> After exploring the binarization modes of different syntax elements and their occurring frequency during the decoding process, we find there are two classes of syntax elements. The syntax elements in the first class contain a few bins and occur only once or a few times in one MB; while the syntax elements in the second class contain several bins and occur tens or hundreds of times in one MB. Syntax elements of the first class contribute only a small part to the arithmetic decoding workload, so in our architecture, one bin is processed per cycle for such kind of syntax elements to decrease the complexity and hardware cost. However to accelerate decoding of the syntax elements in the second class, a new architecture that can decode two regular bins together with one bypass bin in one cycle is proposed. Those syntax elements in the second class include abs_mvd, significant_coeff_flag, last_significant_coeff_flag and coeff_abs_level_minus1. Yu at § III.B.
---	--

¹ For the convenience and clarity, the corrections from the certificate of correction have been incorporated into this limitation and the corresponding limitations of claims 8 and 15, below.

Based on those characteristics of CABAC described above, we organize the elementary engines for one bin as following: 1) a two-hierarchy decoding tree for decoding two regular bins *RB1* and *RB2* with or without a sign, which is illustrated as Tree-(a) in Fig. 3; and 2) a two-hierarchy decoding tree for decoding two bypass bins *BP1* and *BP2*, which is illustrated as Tree-(b) in Fig. 3.

The following paragraphs explain how Tree-(a) and Tree-(b) are employed to decode all syntax elements.

All syntax elements are decoded one regular bin per cycle, utilizing the first branch of the Tree-(a), except for *abs_mvd*, *coeff_abs_level_minus1*, *significant_coeff_flag*, and *last_significant_coeff_flag*.

Yu at § III.B.

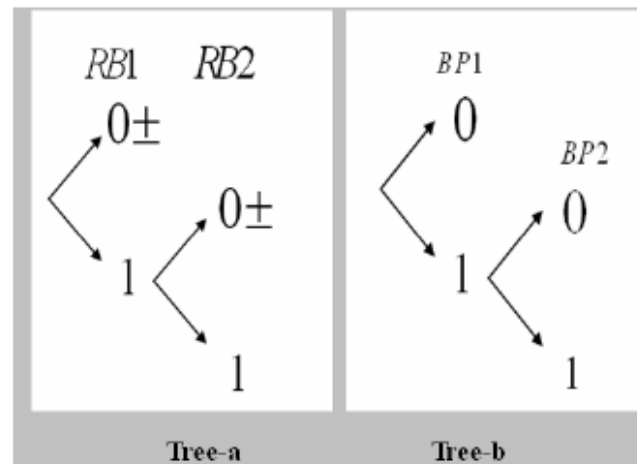


Fig. 3. Two hierarchy decoding tree

Yu at Fig. 3.

	In our architecture, we propose the new and efficient basic decoding circuit units which can decode 1~3 bins per cycle. The basic units are shared by all syntax elements and hence greatly enhance the decoding speed as it takes advantage of the characters of syntax elements. To the best of our knowledge, other arithmetic encoding/decoding architectures averagely encode/decode at most one bin per cycle and they do not take the characteristics of syntax elements into consideration [5], [9], [13]. Yu at § V.
[c] entropy coding symbols that correspond to the first category of syntax elements and that have been subjected to a context update; and	Yu teaches and/or discloses entropy coding symbols that correspond to the first category of syntax elements and that have been subjected to a context update: <i>See, e.g.,</i> Multi-context model is the crucial technique used in CABAC to achieve high compression ratio. In CABAC, different syntax elements are allocated with different context models, and there are totally 399 context models. For most of the cases, one context model is assigned to specific bin of a syntax element or several consecutive bins of a syntax element; but for some special cases, several context models are assigned to the same bin of a syntax element to increase the compression efficiency. For the special cases mentioned above, one correct context model has to be chosen from several context models, based on the information of the same syntax elements of the adjacent left and top MB/block. After the correct context model is chosen, it must be loaded and updated to keep track of the probability estimation of the

	corresponding bin during the decoding process. To keep up with the high decoding speed of the basic decoding circuit units described in the previous section, i.e. one or more bins are decoded per cycle and the bin string of a syntax element is decoded without interruption, the context models must be accessed, updated and saved at the same speed. So, it's not practical to load the context model from RAM and write the renewed value back to RAM during decoding each bin, because at least two cycles are needed to maintain one context model, which is a considerable speed reduction. Context models may be updated simultaneously if they are stored in registers. However, it will cost larger area non-economically if all context models are stored in registers, because totally 399 registers of seven bits are needed. And there'll be more cost for control of those registers. Yu at § III.C.
[d] entropy coding symbols that correspond to the second category of syntax elements and that have bypassed context updating.	Yu teaches and/or discloses entropy coding symbols that correspond to the second category of syntax elements and that have bypassed context updating. <i>See, e.g.,</i>

2) *regbinnum* (1 bit): this signal cooperates with the *dec_mode* to decide how many regular bins are to be decoded in one cycle, only when *dec_mode* is '00', this signal is useful. When decoding prefix of *abs_mvd*, *coeff_abs_level_minus1*, *significant_coeff_flag*, and *last_significant_coeff_flag*, *regbinnum* is 0 indicating two branches of Tree-(a) are both working. When decoding the other regular bins, *regbinnum* is 1 indicating only the first branch of Tree-(a) is working.

3) *rb2_state_1_ns* (1 bit): this signal indicates if decoding the second regular bin uses the renewed context model of the first regular bin or another context model. This signal is useful only when Tree-(a) is in use.

4) *sign1_exist* (1 bit), *sign2_exist* (1 bit): these two signals are useful only when Tree-a is in use. They decide whether to decode a bypass bin as sign after the regular bin(s) are decoded. They are used when decoding *abs_mvd* and *coeff_abs_level_minus1*. *Coeff_abs_level_minus1* contributes most to the decoding workload for I frames. *Coeff_abs_level_minus1* and *abs_mvd* contribute most to the decoding workload for P/B frames. Statistical data shows that small value less than 10 of these two syntax elements accounts for up to 99% of total. Small value less than 6 accounts for about 90% of total. So, with the sign decoded with those regular bins in the same cycle, decoding speed is greatly enhanced. We can decode those syntax elements of value less than 2 in one cycle, of value less than 4 in two cycles, of value less than 6 in three cycles. And this can be achieved without bringing extra cost or prolonging the critical path.

Yu at § IV.A.

2) syntax element `coeff_abs_level_minus1` in 4x4 blocks of *ctxBlockCat* of value 1 (*ctxBlockCat* is a classification label of blocks. *ctxBlockCat* of value 1 corresponds to luma AC transform coefficient levels of MB encoded in Intra_16x16 prediction mode) uses 10 different context models. But this syntax element may appear many times during the process of decoding one MB, and they are interleaved with other three syntax elements `coded_block_flag`, `significant_coeff_flag`, `last_significant_coeff_flag`. To avoid loading and writing back the context models too frequently during the decoding process, the context models for `coded_block_flag`, `significant_coeff_flag`, `last_significant_coeff_flag` and `coeff_abs_level_minus1` of one MB of a certain *ctxBlockCat* are gathered in one group. In this way, this group of at most 44 context models is loaded at the start of decoding the residual block of a certain *ctxBlockCat*, and written back after all residual blocks of the same *ctxBlockCat* are decoded out.

Thus, all 399 context models correspond to certain syntax elements are divided into 18 groups as following (we show groups of syntax elements, all context models corresponding to the bins of the syntax elements in the same group are grouped together):

	1)mb_skip_flag; 2)mb_field_decoding_flag; 3)mb_type for SI slice; 4)mb_type for I slice; 5)mb_type for P slice; 6)mb_type for B slice 7)sub_mb_type for P slice; 8)sub_mb_type for B slice; 9)syntax elements for intra prediction mode, including prev_intra4x4_pred_mode_flag, rem_intra4x4_pred_mode, intra_chroma_pred_mode; 10)ref_idx; 11)abs_mvd; 12) coded_block_pattern; 13)mb_dq_delta; 14)syntax elements for decoding residual data of a certain <i>ctxBlockCat</i>, including coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, coeff_abs_level_minus1. There are five different values of <i>ctxBlockCat</i>, and each of them corresponds to one group. Only the group of the last class described above contains at most 44 context models, other groups contains at most 14 context models. So forty-four 7bits register is enough. In our architecture, all 399 context models are stored in a 32×105 bits single-port RAM. Each row contains 15 context models with some vacant space. This ensures all groups of context models other than that of the last class are placed in the same row, and thus can be loaded and written back in one cycle, and the group of the last class can be loaded and written back within three cycles. Yu at § IV.B.
3. A method according to claim 1 further comprising performing binarization of the symbols that correspond to the second category of syntax elements prior to bypassing the context updating.	Yu teaches and/or discloses a method according to claim 1 further comprising performing binarization of the symbols that correspond to the second category of syntax elements prior to bypassing the context updating. <i>See, e.g.,</i>

This section covers the basic principles of CABAC encoding and decoding process. And we briefly introduce the bottleneck of CABAC encoder/decoder implementation.

The CABAC encoding process consists of three elementary steps [10]:

1) binarization, which is the unique mapping process of possible values of a syntax element onto a set of bin strings. The syntax element is an element of data represented in the bit stream to be encoded. The bin string is a string of bins, which is an intermediate binary representation of value of syntax elements. Each syntax element is uniquely mapped to a bin string and each bin of the bin string is arithmetically encoded into the bit stream.

2) probability modeling. There are totally 399 context models used in CABAC. Context models are used for the probability estimation, each of which corresponds to a certain bin or several bins of a certain syntax element. Sometimes several context models correspond to the same bin. Context models keeps updating during the whole encoding process to keep track of the probability estimation. Before arithmetically encoding one bin, the corresponding context model is selected for the bin and the model is updated after the bin is encoded. Each context model comprises of a pair of two values, a 6-bit state value *state* as the probability index, and a binary value of the most probable symbol *MPS*. The state value is used as an index of the estimated probability value of the least probable symbol "*LPS*" (*LPS* = !*MPS*).

Yu at § II.

	Actually in CABAC arithmetic encoding implementation, two sub-engines are used: one for bins of regular coding mode as described above, and another so-called bypass coding engine for a fast encoding of bins. For such bins, a default value of 0.5 is used as the estimation of p_{LPS}. The two sorts of bins are referred to as “regular bin” and “bypass bin” in the following text. For more detailed description of CABAC, please refer to [10]-[11]. The CABAC decoding process is the inverse process of CABAC encoding. First, the corresponding context model is selected for the bin to be decoded. Then, the bin is decoded by binary arithmetic decoding engine. During the decoding process, the context model should be updated. Binary arithmetic decoding engine is similar to that of binary arithmetic encoding. The current interval is kept in the 9 bits register <i>range</i>, and the position of the input bit stream is kept in the 9 bits register <i>offset</i>. Whether the bin is decoded as <i>MPS</i> or <i>LPS</i> is decided by whether the offset falls into the subinterval of $rLPS$ or $rMPS$. Those two registers are iteratively updated. Binary arithmetic decoding engine is further discussed in more details in section III-A. Finally, bins decoded are post processed to reconstruct the syntax element for output. Yu at § II.
4. A method according to claim 3 wherein performing binarization comprises performing binarization so as to target a predetermined probability.	Yu teaches and/or discloses a method according to claim 3 wherein performing binarization comprises performing binarization so as to target a predetermined probability. See, e.g.,

	3) Binary arithmetic coding. The original principle of the binary arithmetic coding is based on recursive interval subdivision of the interval width R [12]. Given the estimation of the probability p_{LPS} of LPS, the interval is subdivided into two subintervals: one interval width $rLPS = R \cdot p_{LPS}$ which is associated with the LPS, and the other interval width $rMPS = R - rLPS$, which is assigned to the MPS. Depending on whether the observed bin to be encoded is MPS or LPS, the corresponding subinterval is chosen as the new interval. The binary arithmetic coding process keeps updating two registers, i.e. the interval width register R which marks the range of the interval and the code register C which marks the lower bound of the interval. Yu at §II Actually in CABAC arithmetic encoding implementation, two sub-engines are used: one for bins of regular coding mode as described above, and another so-called bypass coding engine for a fast encoding of bins. For such bins, a default value of 0.5 is used as the estimation of p_{LPS}. The two sorts of bins are referred to as “regular bin” and “bypass bin” in the following text. For more detailed description of CABAC, please refer to [10]-[11]. Yu at § II.
6. A method according to claim 1 wherein categorizing the plurality of syntax elements comprises categorizing the plurality of syntax elements based on a relationship of the frequency of occurrence of the syntax elements to a predefined threshold.	Yu teaches and/or discloses a method according to claim 1 wherein categorizing the plurality of syntax elements comprises categorizing the plurality of syntax elements based on a relationship of the frequency of occurrence of the syntax elements to a predefined threshold. See, e.g.,

	After exploring the binarization modes of different syntax elements and their occurring frequency during the decoding process, we find there are two classes of syntax elements. The syntax elements in the first class contain a few bins and occur only once or a few times in one MB; while the syntax elements in the second class contain several bins and occur tens or hundreds of times in one MB. Syntax elements of the first class contribute only a small part to the arithmetic decoding workload, so in our architecture, one bin is processed per cycle for such kind of syntax elements to decrease the complexity and hardware cost. However to accelerate decoding of the syntax elements in the second class, a new architecture that can decode two regular bins together with one bypass bin in one cycle is proposed. Those syntax elements in the second class include <code>abs_mvd</code>, <code>significant_coeff_flag</code>, <code>last_significant_coeff_flag</code> and <code>coeff_abs_level_minus1</code>. Yu at § III.B.
7. A method according to claim 1 further comprising performing binarization of the symbols that correspond to the first category of syntax elements prior to entropy coding the symbols that correspond to the first category of syntax elements.	Yu teaches and/or discloses A method according to claim 1 further comprising performing binarization of the symbols that correspond to the first category of syntax elements prior to entropy coding the symbols that correspond to the first category of syntax elements. <i>See, e.g.,</i> This section covers the basic principles of CABAC encoding and decoding process. And we briefly introduce the bottleneck of CABAC encoder/decoder implementation. The CABAC encoding process consists of three elementary steps [10]:

	1)binarization, which is the unique mapping process of possible values of a syntax element onto a set of bin strings. The syntax element is an element of data represented in the bit stream to be encoded. The bin string is a string of bins, which is an intermediate binary representation of value of syntax elements. Each syntax element is uniquely mapped to a bin string and each bin of the bin string is arithmetically encoded into the bit stream. 2) probability modeling. There are totally 399 context models used in CABAC. Context models are used for the probability estimation, each of which corresponds to a certain bin or several bins of a certain syntax element. Sometimes several context models correspond to the same bin. Context models keeps updating during the whole encoding process to keep track of the probability estimation. Before arithmetically encoding one bin, the corresponding context model is selected for the bin and the model is updated after the bin is encoded. Each context model comprises of a pair of two values, a 6-bit state value <i>state</i> as the probability index, and a binary value of the most probable symbol <i>MPS</i>. The state value is used as an index of the estimated probability value of the least probable symbol “<i>LPS</i>” (<i>LPS</i> = !<i>MPS</i>). Yu at § II.
8. [pre] An apparatus comprising at least one processor and at least one memory including computer program code, the at least one memory and the computer program code configured to, with the processor, cause the apparatus to at least:	See claim 1 [pre], above which is incorporated herein by reference. See, e.g.,

Abstract — *In this paper, we propose a high performance hardware architecture of CABAC decoder. CABAC is the Context Adaptive Binary Arithmetic Coding used in H.264/AVC video standard, which achieves significant compression enhancement while bringing greater complexity and costs in implementation. The necessity of hardware implementation for real-time CABAC decoders is introduced, and then a fast and cost effective architecture is proposed. The new architecture can achieve decoding speed of averagely 500cycles/macroblock, for typical 4M bit stream of D1 resolution, 30frame/s. An ASIC implementation of the new architecture is carried out in a 0.18 μm silicon technology. The estimated area is 0.3mm² and the critical path is limited within 6.7ns¹.*

Yu at Abstract.

In this work, we present a highly efficient architecture for CABAC decoding that takes advantage of the new characters found in CABAC. In this architecture, the whole decoding process inside one MB is implemented by hardware, and software communicates with hardware above the MB layer. The two most time consuming parts of CABAC decoder, i.e. binary arithmetic decoding process and context model maintenance, are carefully designed to make good tradeoff between decoding efficiency and chip cost. Basic arithmetic decoding circuit units are designed to be efficiently shared by all syntax elements. Context models are carefully organized to make sure that during decoding one MB, interruptions for loading context models from RAM and writing back the renewed context models to RAM are kept at the lowest rate.

Yu at § I.

Our architecture deals with the whole decoding procedure of MB by hardware. To the best of our knowledge, no other hardware implementations designed for CABAC coder/decoder is based on one MB. The best of the architecture proposed till now is the arithmetic coding architecture proposed by Osorio et al. [5], where only `significant_coeff_flag`, `last_significant_coeff_flag` and `coeff_abs_level_minus1` are accelerated by hardware, with much more complicated control and interaction cost between software and hardware.

In our architecture, we propose the new and efficient basic decoding circuit units which can decode 1~3 bins per cycle. The basic units are shared by all syntax elements and hence greatly enhance the decoding speed as it takes advantage of the characters of syntax elements. To the best of our knowledge, other arithmetic encoding/decoding architectures averagely encode/decode at most one bin per cycle and they do not take the characteristics of syntax elements into consideration [5], [9], [13].

We also propose an efficient context managing mechanism which divides context models into several groups according to the order of being called. This guarantees that loading and writing context models cost only a few cycles during decoding one MB. Only 44 registers of seven bits are needed to keep updating the context models. Some caching and pre-fetching mechanism is proposed to manage context models in earlier papers [5], [7]. However, their control is much more complicated and the whole decoding speed is not as fast as ours.

Yu at § V.

[a] determine a frequency of occurrence threshold based on an expected frequency of occurrence of syntax elements in a bit stream;	See limitation 1[a], above which is incorporated herein by reference.
[b] categorize a plurality of syntax elements of video content into first and second categories based on the frequency of occurrence threshold, wherein syntax elements which occur greater than the frequency of occurrence threshold are categorized into the second category and syntax elements which occur less than the frequency of occurrence are categorized into the first category;	See limitation 1[b], above which is incorporated herein by reference.
[c] entropy code symbols that correspond to the first category of syntax elements and that have been subjected to a context update; and	See limitation 1[c], above which is incorporated herein by reference.
[d] entropy code symbols that correspond to the second category of syntax elements and that have bypassed context updating.	See limitation 1[d], above which is incorporated herein by reference.
10. An apparatus according to claim 8 wherein the at least one memory and the computer program code are further configured to, with the processor, cause the apparatus to perform binarization of the symbols that correspond to the second category of syntax elements	See claim 3, above which is incorporated herein by reference.

prior to bypassing the context updating.	
11. An apparatus according to claim 10 wherein the at least one memory and the computer program code are configured to, with the processor, cause the apparatus to perform binarization by performing binarization so as to target a predetermined probability.	See claim 4, above which is incorporated herein by reference.
13. An apparatus according to claim 8 wherein the at least one memory and the computer program code are configured to, with the processor, cause the apparatus to categorize the plurality of syntax elements by categorizing the plurality of syntax elements based on a relationship of the frequency of occurrence of the syntax elements to a predefined threshold.	See claim 6, above which is incorporated herein by reference.

14. An apparatus according to claim 8 wherein the at least one memory and the computer program code are further configured to, with the processor, cause the apparatus to perform binarization of the symbols that correspond to the first category of syntax elements prior to entropy coding the symbols that correspond to the first category of syntax elements.	See claim 7, above which is incorporated herein by reference.
15. [pre] A computer program product comprising at least one non-transitory computer-readable storage medium having computer-executable program code instructions stored therein, the computer-executable program code instructions comprising program code instructions to:	See claim 1 [pre], above which is incorporated herein by reference. See, e.g., <i>Abstract — In this paper, we propose a high performance hardware architecture of CABAC decoder. CABAC is the Context Adaptive Binary Arithmetic Coding used in H.264/AVC video standard, which achieves significant compression enhancement while bringing greater complexity and costs in implementation. The necessity of hardware implementation for real-time CABAC decoders is introduced, and then a fast and cost effective architecture is proposed. The new architecture can achieve decoding speed of averagely 500cycles/macroblock, for typical 4M bit stream of D1 resolution, 30frame/s. An ASIC implementation of the new architecture is carried out in a 0.18 μm silicon technology. The estimated area is 0.3mm² and the critical path is limited within 6.7ns¹.</i> Yu at Abstract.

In nowadays, a large number of consumer products such as digital cameras, Personal Digital Assistants, video telephony, portable DVD player as well as storage, broadcast and streaming of standard definition TV are common practice. All those applications demand efficient management of the large amount of video data. This motivated a large body of research in industry as well as in academia to develop advanced video coding technology. H.264/AVC [1] video coding standard developed by the ITU-T/ISO/IEC is the latest international standard developed by ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group. The new standard provides gains in compression efficiency of up to 50% over a wide range of bit rates and video resolutions compared with the former standards. However, the decoder complexity is about four times that of MPEG-2 and two times of MPEG-4 Visual Simple Profile. Some of the major tools in H.264/AVC contributing to the computational complexity include variable block sizes, Hadamard transform, RD-Lagrangian optimization, B-frames, 1/4 displacement vector resolution, increasing search range and reference frame number, and CABAC [4].

Yu at § I.

In our architecture, we propose the new and efficient basic decoding circuit units which can decode 1~3 bins per cycle. The basic units are shared by all syntax elements and hence greatly enhance the decoding speed as it takes advantage of the characters of syntax elements. To the best of our knowledge, other arithmetic encoding/decoding architectures averagely encode/decode at most one bin per cycle and they do not take the characteristics of syntax elements into consideration [5], [9], [13].

We also propose an efficient context managing mechanism which divides context models into several groups according to the order of being called. This guarantees that loading and writing context models cost only a few cycles during decoding one MB. Only 44 registers of seven bits are needed to keep updating the context models. Some caching and pre-fetching mechanism is proposed to manage context models in earlier papers [5], [7]. However, their control is much more complicated and the whole decoding speed is not as fast as ours.

We have implemented this architecture in verilog using 0.18 μm technology. Synthetic results show that the total area is about 0.3mm^2 with critical path limited within 6.7ns. This result does not count in the 32×105 bits RAM memory for storing the 399 context models. We conducted tests on a typical 4M/s bit stream of D1 resolution, and results show that averagely one MB can be decoded within 500 cycles. This is quite rich for D1 resolution real-time application. Much space is left for real-time applications of higher video quality and larger image resolutions in the future.

	Yu at § V.
[a] determine a frequency of occurrence threshold based on an expected frequency of occurrence of syntax elements in a bit stream;	See limitation 1[a], above which is incorporated herein by reference.
[b] categorize a plurality of syntax elements of video content into first and second categories based on the frequency of occurrence threshold, wherein syntax elements which occur greater than the frequency of occurrence threshold are categorized into the second category and syntax elements which occur less than the frequency of occurrence are categorized into the first category;	See limitation 1[b], above which is incorporated herein by reference.
[c] entropy code symbols that correspond to the first category of syntax elements and that have been subjected to a context update; and	See limitation 1[c], above which is incorporated herein by reference.
[d] entropy code symbols that correspond to the second category of syntax elements and that have bypassed context updating.	See limitation 1[d], above which is incorporated herein by reference.

17. A computer program product according to claim 15 wherein the computer-executable program code instructions further comprise program code instructions to perform binarization of the symbols that correspond to the second category of syntax elements prior to bypassing the context updating.	See claim 3, above which is incorporated herein by reference.
18. A computer program product according to claim 17 wherein the comprise program code instructions to perform binarization comprise program code instructions to perform binarization so as to target a predetermined probability.	See claim 4, above which is incorporated herein by reference.
20. A computer program product according to claim 15 comprise the program code instructions to categorize the plurality of syntax elements comprise program code instructions to categorize the plurality of syntax elements based on a relationship of the frequency of occurrence of the syntax elements to a predefined threshold.	See claim 6, above which is incorporated herein by reference.