Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard

Detlev Marpe, *Member, IEEE*, Heiko Schwarz, and Thomas Wiegand

***Abstract*—Context-Based Adaptive Binary Arithmetic Coding (CABAC) as a normative part of the new ITU-T/ISO/IEC standard H.264/AVC for video compression is presented. By combining an adaptive binary arithmetic coding technique with context modeling, a high degree of adaptation and redundancy reduction is achieved. The CABAC framework also includes a novel low-complexity method for binary arithmetic coding and probability estimation that is well suited for efficient hardware and software implementations. CABAC significantly outperforms the baseline entropy coding method of H.264/AVC for the typical area of envisaged target applications. For a set of test sequences representing typical material used in broadcast applications and for a range of acceptable video quality of about 30 to 38 dB, average bit-rate savings of 9%–14% are achieved.**

***Index Terms*—Binary arithmetic coding, CABAC, context modeling, entropy coding, H.264, MPEG-4 AVC.**

I. Introduction

NATURAL camera-view video signals show nonstationary statistical behavior. The statistics of these signals largely depend on the video content and the acquisition process. Traditional concepts of video coding that rely on a mapping from the video signal to a bitstream of variable length-coded syntax elements exploit some of the nonstationary characteristics but certainly not all of it. Moreover, higher order statistical dependencies on a syntax element level are mostly neglected in existing video coding schemes. Designing an entropy coding scheme for a video coder by taking into consideration these typically observed statistical properties, however, offers room for significant improvements in coding efficiency.

Context-Based Adaptive Binary Arithmetic Coding (CABAC) is one of the two entropy coding methods of the new ITU-T/ISO/IEC standard for video coding, H.264/AVC [1], [2]. The algorithm was first introduced in a rudimentary form in [7] and evolved over a period of successive refinements [8]–[17]. In this paper, we present a description of the main elements of the CABAC algorithm in its final standardized form as specified in [1]. Unlike the specification in [1], the presentation in this paper is intended to provide also some information on the underlying conceptual ideas as well as the theoretical and historical background of CABAC.

Entropy coding in today's hybrid block-based video coding standards such as MPEG-2 [3], H.263 [4], and MPEG-4 [5] is generally based on fixed tables of variable-length codes (VLCs). For coding the residual data in these video coding standards, a block of transform coefficient levels is first mapped onto a one-dimensional list using a pre-defined scanning pattern. This list of transform coefficient levels is then coded using a combination of run-length and variable length coding. Due to the usage of VLCs, coding events with a probability greater than 0.5 cannot be efficiently represented, and hence, a so-called *alphabet extension* of "run" symbols representing successive levels with value zero is used in the entropy coding schemes of MPEG-2, H.263, and MPEG-4. Moreover, the usage of fixed VLC tables does not allow an adaptation to the actual symbol statistics, which may vary over space and time as well as for different source material and coding conditions. Finally, since there is a fixed assignment of VLC tables and syntax elements, existing inter-symbol redundancies cannot be exploited within these coding schemes.

Although, from a conceptual point-of-view, it is well known for a long time that all these deficiencies can be most easily resolved by *arithmetic codes* [23], little of this knowledge was actually translated into practical entropy coding schemes specifically designed for block-based hybrid video coding. One of the first hybrid block-based video coding schemes that incorporate an adaptive binary arithmetic coder capable of adapting the model probabilities to the existing symbol statistics was presented in [6]. The core of that entropy coding scheme was inherited from the JPEG standard (at least for coding of DCT coefficients) [25], and an adjustment of its modeling part to the specific statistical characteristics of typically observed residual data in a hybrid video coder was not carried out. As a result, the performance of this JPEG-like arithmetic entropy coder in the hybrid block-based video coding scheme of [6] was not substantially better for inter-coded pictures than that of its VLC-based counterpart.

The first and—until H.264/AVC was officially released—the only standardized arithmetic entropy coder within a hybrid block-based video coder is given by Annex E of H.263 [4]. Three major drawbacks in the design of that optional arithmetic coding scheme can be identified. First, Annex E is applied to the same syntax elements as the VLC method of H.263 including the combined symbols for coding of transform coefficient levels. Thus, one of the fundamental advantages of arithmetic coding that a noninteger code length can be assigned to coding events is unlikely to be exploited. Second, all probability models in Annex E of H.263 are nonadaptive in the sense that their underlying probability distributions are assumed to be static. Although multiple probability distribution models are defined and chosen in a frequency-dependent way for the combined symbols of run, level and "last" information,

Manuscript received April 10, 2003; revised May 6, 2003.

The authors are with the Fraunhofer-Institute for Telecommunications – Heinrich-Hertz Institute (HHI), 10587 Berlin, Germany.

Digital Object Identifier 10.1109/TCSVT.2003.815173

this conditioning does not result in a significant gain in coding efficiency, since an adaptation to the actual symbol statistics is not possible. Finally, the generic m-ary arithmetic coder used in Annex E involves a considerable amount of computational complexity, which may not be justified in most application scenarios, especially in view of the typically observed small margins of coding gains.

Entropy coding schemes based on arithmetic coding are quite frequently involved in the field of non block-based video coding. Most of these alternative approaches to video coding are based on the discrete wavelet transform (DWT) in combination with disparate methods of temporal prediction, such as overlapped block motion compensation, grid-based warping, or motion-compensated temporal filtering [18]–[20]. The corresponding entropy coding schemes are often derived from DWT-based still image coding schemes like SPIHT [21] or other predecessors of JPEG2000 [35].

In our prior work on wavelet-based hybrid video coding, which led to one of the proposals for the H.26L standardization [19], the entropy coding method of *partitioning, aggregation and conditional coding* (PACC) was developed [22]. One of its main distinguishing features is related to the partitioning strategy: Given a source with a specific alphabet size, for instance, quantized transform coefficients, it was found to be useful to first reduce the alphabet size by partitioning the range according to a binary selector which, e.g., in the case of transform coefficients, would be typically given by the decision whether the coefficient is quantized to zero or not. In fact, range partitioning using binary selectors can be viewed as a special case of a *binarization scheme*, where a symbol of a nonbinary alphabet is uniquely mapped to a sequence of binary decisions prior to further processing.

This (somehow) dual operation to the aforementioned alphabet extension, which in the sequel we will therefore refer to as *alphabet reduction*, is mainly motivated by the fact that it allows the subsequent modeling stage to operate more efficiently on this maximally reduced (binary) alphabet. In this way, the design and application of higher order conditioning models is greatly simplified and, moreover, the risk of "overfitting" the model is reduced. As a positive side effect, a fast table-driven binary arithmetic coder can be utilized for the final arithmetic coding stage.

The design of CABAC is in the spirit of our prior work. To circumvent the drawbacks of the known entropy coding schemes for hybrid block-based video coding such as Annex E of H.263, we combine an adaptive binary arithmetic coding technique with a well-designed set of context models. Guided by the principle of alphabet reduction, an additional binarization stage is employed for all nonbinary valued symbols. Since the increased computational complexity of arithmetic coding in comparison to variable length coding is generally considered as its main disadvantage, great importance has been devoted to the development of an algorithmic design that allows efficient hardware and software implementations.

For some applications, however, the computational requirements of CABAC may be still too high given today's silicon technology. Therefore, the baseline entropy coding method of H.264/AVC [1] offers a different compression-complexity tradeoff operating at a reduced coding efficiency and complexity level compared to CABAC. It mostly relies on a single infinite-extended codeword set consisting of zero-order Exp-Golomb codes, which are used for all syntax elements except for the residual data. For coding the residual data, a more sophisticated method called *Context-Adaptive Variable-Length Coding* (CAVLC) is employed. In this scheme, inter-symbol redundancies are exploited by switching VLC tables for various syntax elements depending on already transmitted coding symbols [1], [2]. The CAVLC method, however, cannot provide an adaptation to the actually given conditional symbol statistics. Furthermore, coding events with symbol probabilities greater than 0.5 cannot be efficiently coded due to the fundamental lower limit of 1 bit/symbol imposed on variable length codes. This restriction prevents the usage of coding symbols with a smaller alphabet size for coding the residual data, which could allow a more suitable construction of contexts for switching between the model probability distributions.

The remainder of the paper is organized as follows. In Section II, we present an overview of the CABAC framework including a high-level description of its three basic building blocks of binarization, context modeling and binary arithmetic coding. We also briefly discuss the motivation and the principles behind the algorithmic design of CABAC. A more detailed description of CABAC is given in Section III, where the individual steps of the algorithm are presented in depth. Finally, in Section IV we provide experimental results to demonstrate the performance gains of CABAC relative to the baseline entropy coding mode of H.264/AVC for a set of interlaced video test sequences.

II. The CABAC Framework

Fig. 1 shows the generic block diagram for encoding a single syntax element in CABAC.[1] The encoding process consists of, at most, three elementary steps:

1) binarization;
2) context modeling;
3) binary arithmetic coding.

In the first step, a given nonbinary valued syntax element is uniquely mapped to a binary sequence, a so-called *bin string*. When a binary valued syntax element is given, this initial step is bypassed, as shown in Fig. 1. For each element of the bin string or for each binary valued syntax element, one or two subsequent steps may follow depending on the coding mode.

In the so-called *regular coding mode*, prior to the actual arithmetic coding process the given binary decision which, in the sequel, we will refer to as a *bin*, enters the context modeling stage, where a probability model is selected such that the corresponding choice may depend on previously encoded syntax elements or bins. Then, after the assignment of a context model the bin value along with its associated model is passed to the regular coding engine, where the final stage of arithmetic encoding together with a subsequent model updating takes place (see Fig. 1).

[1]For simplicity and for clarity of presentation, we restrict our exposition of CABAC to an encoder only view. In the text of the H.264/AVC standard [1] itself, the converse perspective dominates—the standard normatively specifies only how to decode the video content without specifying how to encode it.

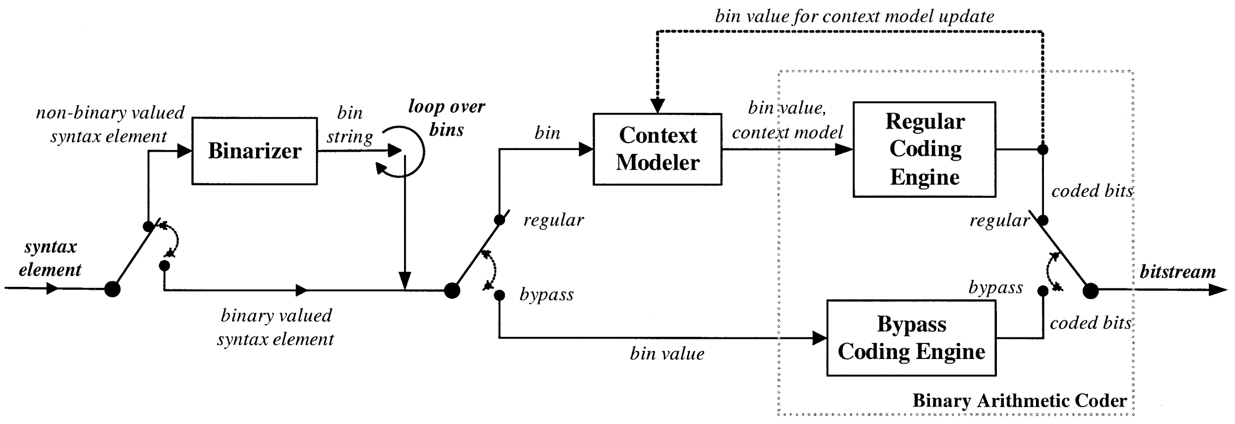


Fig. 1. CABAC encoder block diagram.

Alternatively, the *bypass coding mode* is chosen for selected bins in order to allow a speedup of the whole encoding (and decoding) process by means of a simplified coding engine without the usage of an explicitly assigned model, as illustrated by the lower right branch of the switch in Fig. 1.

In the following, the three main functional building blocks, which are binarization, context modeling, and binary arithmetic coding, along with their interdependencies are discussed in more detail.

A. Binarization

1) General Approach: For a successful application of context modeling and adaptive arithmetic coding in video coding we found that the following two requirements should be fulfilled:

a) a fast and accurate estimation of conditional probabilities must be achieved in the relatively short time interval of a slice coding unit;
b) the computational complexity involved in performing each elementary operation of probability estimation and subsequent arithmetic coding must be kept at a minimum to facilitate a sufficiently high throughput of these inherently sequentially organized processes.

To fulfill both requirements we introduce the important "pre-processing" step of first reducing the alphabet size of the syntax elements to encode. Alphabet reduction in CABAC is performed by the application of a binarization scheme to each nonbinary 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.

First, it is important to note that nothing is lost in terms of modeling, since the individual (nonbinary) symbol probabilities can be recovered by using the probabilities of the individual bins of the bin string. For illustrating this aspect, let us consider the binarization for the syntax element mb_type of a P/SP slice.

As depicted in Fig. 2(a), the terminal nodes of the binary tree correspond to the symbol values of the syntax element such that the concatenation of the binary decisions for traversing the tree from the root node to the corresponding terminal node represents the bin string of the corresponding symbol value. For instance, consider the value "3" of mb_type, which signals the

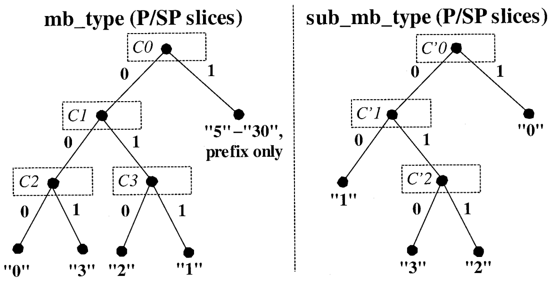


Fig. 2. Illustration of the binarization for (a) mb_type and (b) sub_mb_type both for P/SP slices.

macroblock type "P_8×8", i.e., the partition of the macroblock into four 8×8 submacroblocks in a P/SP slice. In this case, the corresponding bin string is given by "001". As an obvious consequence, the symbol probability $p(\text{"3"})$ is equal to the product of the probabilities $p^{(C0)}(\text{"0"})$, $p^{(C1)}(\text{"0"})$, and $p^{(C2)}(\text{"1"})$, where $C0$, $C1$ and $C2$ denote the (binary) probability models of the corresponding internal nodes, as shown in Fig. 2. This relation is true for any symbol represented by any such binary tree, which can be deduced by the iterated application of the Total Probability Theorem [26].

Although at this stage nothing seems to be gained, there is already the advantage of using a binary arithmetic coding engine on the bin string instead of a m-ary arithmetic coder operating on the original m-ary source alphabet. Adaptive m-ary arithmetic coding (for $m > 2$) is in general a computationally complex operation requiring at least two multiplications for each symbol to encode as well as a number of fairly complex operations to perform the update of the probability estimation [36]. In contrast to that, there are fast, multiplication-free variants of binary arithmetic coding, one of which was specifically developed for the CABAC framework, as further described below. Since the probability of symbols with larger bin strings is typically very low, the computational overhead of coding all bins of that bin string instead of using only one pass in an m-ary arithmetic coder is fairly small and can be easily compensated by using a fast binary coding engine.

Finally, as the most important advantage, binarization enables context modeling on a subsymbol level. For specific bins which, in general, are represented by the most frequently observed bins, conditional probabilities can be used, whereas other usually less frequently observed bins can be treated using a joint, typically zero-order, probability model. Compared to the conventional approach of using context models in the original domain of the source with typically large alphabet size (like e.g., components of motion vector differences or transform coefficient levels) this additional freedom in the design offers a flexible instrument for using higher order conditional probabilities without suffering from context "dilution" effects. These effects are often observed in cases, where a large number of conditional probabilities have to be adaptively estimated on a relatively small (coding) time interval, such that there are not enough samples to reach a reliable estimate for each model.[2]

For instance, when operating in the original alphabet domain, a quite moderately chosen second-order model for a given syntax element alphabet of size $m = 256$ will result in the intractably large number of $256^2 \cdot (256 - 1) \approx 2^{24}$ symbol probabilities to be estimated for that particular syntax element only. Even for a zero-order model, the task of tracking 255 individual probability estimates according to the previous example is quite demanding. However, typically measured probability density functions (pdf) of prediction residuals or transformed prediction errors can be modeled by highly peaked Laplacian, generalized Gaussian or geometric distributions [28], where it is reasonable to restrict the estimation of individual symbol statistics to the area of the largest statistical variations at the peak of the pdf. Thus, if, for instance, a binary tree resulting from a Huffman code design would be chosen as a binarization for such a source and its related pdf, only the nodes located in the vicinity of the root node would be natural candidates for being modeled individually, whereas a joint model would be assigned to all nodes on deeper tree levels corresponding to the "tail" of the pdf. Note that this design is different from the example given in Fig. 2, where each (internal) node has its own model.

In the CABAC framework, typically only the root node would be modeled using higher order conditional probabilities. In the above example of a second-order model, this would result in only four different binary probability models instead of m^2 different m-ary probability models with $m = 256$.

2) Design of CABAC Binarization Schemes: As already indicated above, a binary representation for a given nonbinary valued syntax element provided by the binarization process should be close to a minimum-redundancy code. On the one hand, this allows easy access to the most probable symbols by means of the binary decisions located at or close to the root node for the subsequent modeling stage. On the other hand, such a code tree minimizes the number of binary symbols to encode on the average, hence minimizing the computational workload induced by the binary arithmetic coding stage.

However, instead of choosing a Huffman tree for a given training sequence, the design of binarization schemes in CABAC (mostly) relies on a few basic code trees, whose structure enables a simple on-line computation of all code words without the need for storing any tables. There are four such basic types: the *unary code*, the *truncated unary code*, the *kth order Exp-Golomb code*, and the *fixed-length code*. In addition, there are binarization schemes based on a concatenation of 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. Two examples of such trees are shown in Fig. 2.

[2]A more rigorous treatment of that problem can be found in [23] and [24]

```
while( 1 ) {
   // unary prefix part of EGk
   if ( x >= (1<<k) ) {
      put( 1 )
      x = x - (1<<k)
      k++
   } else {
      put( 0 ) // terminating "0" of prefix part
      while( k-- ) // binary suffix part of EGk
        put( (x>>k) & 0x01 )
      break
   }
}
```

Fig. 3. Construction of kth order EGk code for a given unsigned integer symbol x.

In the remaining part of this section, we explain in more detail the construction of the four basic types of binarization and its derivatives.

Unary and Truncated Unary Binarization Scheme: For each unsigned integer valued symbol $x \geq 0$, the unary code word in CABAC consists of x "1" bits plus a terminating "0" bit. The truncated unary (TU) code is only defined for x with $0 \leq x \leq S$, where for $x < S$ the code is given by the unary code, whereas for $x = S$ the terminating "0" bit is neglected such that the TU code of $x = S$ is given by a codeword consisting of x "1" bits only.

kth order Exp-Golomb Binarization Scheme: Exponential Golomb codes were first proposed by Teuhola [29] in the context of run-length coding schemes. This parameterized family of codes is a derivative of Golomb codes, which have been proven to be optimal prefix-free codes for geometrically distributed sources [30]. Exp-Golomb codes are constructed by a concatenation of a prefix and a suffix code word. Fig. 3 shows the construction of the kth order Exp-Golomb (EGk) code word for a given unsigned integer symbol x. The prefix part of the EGk code word consists of a unary code corresponding to the value of $l(x) = \lfloor \log_2(x/2^k + 1) \rfloor$.

The EGk suffix part is computed as the binary representation of $x + 2^k(1 - 2^{l(x)})$ using $k + l(x)$ significant bits, as can be seen from the pseudo-C code in Fig. 3.

Consequently, for the EGk binarization, the number of symbols having the same code length of $k + 2 \cdot l(x) + 1$ is geometrically growing. By inverting Shannon's relationship between ideal code length and symbol probability, we can, e.g., easily deduce that EG0 is the optimal code for a pdf $p(x) = 1/2 \cdot (x + 1)^{-2}$ with $x \geq 0$. This implies that for an appropriately chosen parameter k, the EGk code represents a fairly good first-order approximation of the ideal prefix-free code for tails of typically observed pdf's, at least for syntax elements that are representing prediction residuals.

Fixed-Length (FL) Binarization Scheme: For the application of FL binarization, a finite alphabet of values of the corresponding syntax element is assumed. Let x denote a given value of such a syntax element, where $0 \leq x < S$. Then, the

FL code word of x is simply given by the binary representation of x with a fixed (minimum) number $l_{FL} = \lceil \log_2 S \rceil$ of bits. Typically, FL binarization is applied to syntax elements with a nearly uniform distribution or to syntax elements, where each bit in the FL binary representation represents a specific coding decision as e.g., in the part of the coded block pattern symbol related to the luminance residual data.

Concatenation of Basic Binarization Schemes: From the basic binarization schemes as described above, three more binarization schemes are derived. The first one is a concatenation of a 4-bit FL prefix as a representation of the luminance related part of the coded block pattern and a TU suffix with $S = 2$ representing the chrominance related part of coded_block_pattern.

Both the second and third concatenated scheme are derived from the TU and the EGk binarization. These schemes, which are referred to as *Unary/kth order Exp-Golomb* (UEGk) binarizations, are applied to motion vector differences and absolute values of transform coefficient levels. The design of these concatenated binarization schemes is motivated by the following observations. First, the unary code is the simplest prefix-free code in terms of implementation cost. Secondly, it permits a fast adaptation of the individual symbol probabilities in the subsequent context modeling stage, since the arrangement of the nodes in the corresponding tree is typically such that with increasing distance of the internal nodes from the root node the corresponding binary probabilities are less skewed.[3] These observations are only accurate for small values of the absolute motion vector differences and transform coefficient levels. For larger values, there is not much use of an adaptive modeling leading to the idea of concatenating an adapted truncated unary tree as a prefix and a static Exp-Golomb code tree as a suffix. Typically, for larger values, the EGk suffix part represents already a fairly good fit to the observed probability distribution, as already mentioned above. Thus, it is reasonable to speedup the encoding of the bins related to the EGk suffix part in CABAC by using the fast bypass coding engine for uniformly distributed bins, as further described in Section III-D.

For motion vector differences, UEGk binarization is constructed as follows. Let us assume the value mvd of a motion vector component is given. For the prefix part of the UEGk bin string, a TU binarization with a cutoff value of $S = 9$ is invoked for $\min(|mvd|, 9)$. If mvd is equal to zero, the bin string consists only of the prefix code word "0". If the condition $|mvd| \geq 9$ holds, the suffix is constructed as an EG3 codeword for the value of $|mvd| - 9$, to which the sign of mvd is appended using the sign bit "1" for a negative mvd and the sign bit "0" otherwise. For mvd values with $0 < |mvd| < 9$, the suffix consists only of the sign bit. Noting that the component of a motion vector difference represents the prediction error at quarter-sample accuracy, the prefix part always corresponds to a maximum error component of ± 2 samples. With the choice of the Exp-Golomb parameter $k = 3$, the suffix code words are given such that a geometrical increase of the prediction error in units of two samples is captured by a linear increase in the corresponding suffix code word length.

[3]The aspect of a suitable ordering of nodes in binary trees for optimal modeling and fast adaptation has been addressed in [31], although in a slightly different context.

TABLE I
UEG0 BINARIZATION FOR ENCODING OF ABSOLUTE VALUES OF TRANSFORM COEFFICIENT LEVELS

abs_level	Bin string																			
	TU prefix														EG0 suffix					
1	0																			
2	1	0																		
3	1	1	0																	
4	1	1	1	0																
5	1	1	1	1	0															
...	...	...	...	...	...	...														
...	...	...	...	...	...	...	...	...	...	...	...	...								
13	1	1	1	1	1	1	1	1	1	1	1	1	0							
14	1	1	1	1	1	1	1	1	1	1	1	1	1	0						
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0					
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0			
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1			
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	
19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	
20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
bin	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...

UEGk binarization of absolute values of transform coefficient levels (abs_level) is specified by the cutoff value $S = 14$ for the TU prefix part and the order $k = 0$ for the EGk suffix part. Note that the binarization and subsequent coding process is applied to the syntax element $\mathrm{coeff\_abs\_value\_minus1} = \mathrm{abs\_level} - 1$, since zero valued transform coefficient levels are encoded using a significance map, as described in more detail in Section III-B. The construction of a bin string for a given value of coeff_abs_value_minus1 is similar to the construction of UEGk bin strings for the motion vector difference components except that no sign bit is appended to the suffix. Table I shows the corresponding bin strings for values of abs_level from 1 to 20, where the prefix parts are highlighted in gray shaded columns.

B. 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. Since it is the model that determines the code and its efficiency in the first place, it is of paramount importance to design an adequate model that explores the statistical dependencies to a large degree and that this model is kept "up to date" during encoding. However, there are significant model costs involved by adaptively estimating higher order conditional probabilities.

Suppose a pre-defined set $\boldsymbol{T}$ of past symbols, a so-called *context template*, and a related set $\boldsymbol{C} = \{0, \ldots, C-1\}$ of *contexts* is given, where the contexts are specified by a *modeling function* $F: \boldsymbol{T} \rightarrow \boldsymbol{C}$ operating on the template $\boldsymbol{T}$. For each symbol x to be coded, a conditional probability $p(x|F(z))$ is estimated by switching between different probability models according to the already coded neighboring symbols $z \in \boldsymbol{T}$. After encoding x

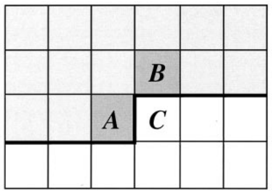


Fig. 4. Illustration of a context template consisting of two neighboring syntax elements A and B to the left and on top of the current syntax element C.

using the estimated conditional probability $p(x|F(z))$, the probability model is updated with the value of the encoded symbol x. Thus, $p(x|F(z))$ is estimated on the fly by tracking the actual source statistics. Since the number τ of different conditional probabilities to be estimated for an alphabet size of m is equal to $\tau = C(m-1)$, it is intuitively clear that the model cost, which represents the cost of "learning" the model distribution, is proportional to τ.[4] This implies that by increasing the number C of different context models, there is a point where overfitting of the model may occur such that inaccurate estimates of $p(x|F(z))$ will be the result.

In CABAC, this problem is solved by imposing two severe restrictions on the choice of the context models. First, very limited context templates $\boldsymbol{T}$ consisting of a few neighbors of the current symbol to encode are employed such that only a small number of different context models C is effectively used. Second, as already motivated in the last section, context modeling is restricted to selected bins of the binarized symbols. As a result, the model cost is drastically reduced, even though the ad-hoc design of context models under these restrictions may not result in the optimal choice with respect to coding efficiency. In fact, in a recently conducted research, it has been shown that additional gains can be obtained by applying the novel GRASP algorithm for an optimized selection of context models using larger context templates within the CABAC framework [31]. However, the improvements are quite moderate compared to the drastic increase in complexity required for performing the two-pass GRASP algorithm.

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. Usually, the specification of this kind of context model for a specific bin is based on a modeling function of the related bin values for the neighboring element to the left and on top of the current syntax element, as shown in Fig. 4.

The second type of context models is only defined for the syntax elements of mb_type and sub_mb_type. For this kind of context models, the values of prior coded bins $(b_0, b_1, b_2, \ldots, b_{i-1})$ are used for the choice of a model for a given bin with index i. Note that in CABAC these context models are only used to select different models for different internal nodes of the corresponding binary trees, as already discussed in Section II-A.

[4]Rissanen derived a refinement of that model cost measure by also taking into account that the precision of estimating the probabilities increases with the number of observations [24]

TABLE II
SYNTAX ELEMENTS AND ASSOCIATED RANGE OF CONTEXT INDICES

Syntax element	Slice type		
	SI/I	P/SP	B
mb_type	0/3-10	14-20	27-35
mb_skip_flag		11-13	24-26
sub_mb_type		21-23	36-39
mvd (horizontal)		40-46	40-46
mvd (vertical)		47-53	47-53
ref_idx		54-59	54-59
mb_qp_delta	60-63	60-63	60-63
intra_chroma_pred_mode	64-67	64-67	64-67
prev_intra4x4_pred_mode_flag	68	68	68
rem_intra4x4_pred_mode	69	69	69
mb_field_decoding_flag	70-72	70-72	70-72
coded_block_pattern	73-84	73-84	73-84
coded_block_flag	85-104	85-104	85-104
significant_coeff_flag	105-165, 277-337	105-165, 277-337	105-165, 277-337
last_significant_coeff_flag	166-226, 338-398	166-226, 338-398	166-226, 338-398
coeff_abs_level_minus1	227-275	227-275	227-275
end_of_slice_flag	276	276	276

Both the third and fourth type of context models is applied to residual data only. In contrast to all other types of context models, both types depend on the context categories of different block types, as specified below. Moreover, 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 (decoded) levels with a specific value prior to the current level bin to encode (decode).

Besides these context models based on conditional probabilities, there are fixed assignments of probability models to bin indices for all those bins that have to be encoded in regular mode and to which no context model of the previous specified categories is applied.

The entity of probability models used in CABAC can be arranged in a linear fashion such that each model can be identified by a unique so-called *context index* γ. Table II contains an overview of the syntax elements in H.264/AVC and its related context indices. The numbering of indices was arranged in such a way that the models related to mb_type, sub_mb_type, and mb_skip_flag for different slice types are distinguished by their corresponding indices. Although the ordering of models in Table II is clearly not the most economical way of housekeeping the models in CABAC, it serves the purpose of demonstrating the conceptual idea.

Each probability model related to a given context index γ is determined by a pair of two values, a 6-bit *probability state index* σ_γ and the (binary) value ϖ_γ of the *most probable symbol* (MPS), as will be further described in Section III-C. Thus, the pairs $(\sigma_\gamma, \varpi_\gamma)$ for $0 \leq \gamma \leq 398$ and hence the models themselves can be efficiently represented by 7-bit unsigned integer values.

The context indices in the range from 0 to 72 are related to syntax elements of macroblock type, submacroblock type, and prediction modes of spatial and of temporal type as well as slice-based and macroblock-based control information. For this

TABLE III
VALUES OF CONTEXT INDEX OFFSET Δ DEPENDING ON CONTEXT CATEGORY (AS SPECIFIED IN TABLE IV) AND SYNTAX ELEMENT

Syntax element	Context category (*ctx_cat*)				
	0	1	2	3	4
coded_block_flag	0	4	8	12	16
significant_coeff_flag	0	15	29	44	47
last_significant_coeff_flag	0	15	29	44	47
coeff_abs_level_minus1	0	10	20	30	39

type of syntax elements, a corresponding context index γ can be calculated as

$$\gamma = \Gamma_S + \chi_S \tag{1}$$

where Γ_S denotes the so-called *context index offset*, which is defined as the lower value of the range given in Table II, and χ_S denotes the *context index increment* of a given syntax element S. Note that the variable χ_S may depend only on the bin index, in which case a fixed assignment of probability model is given, or alternatively, it may specify one of the first or second type of context models, as given above. The notation introduced in (1) will be used in Section III-A for a more detailed description of the context models for syntax elements of the above given type.

Context indices in the range from 73 to 398 are related to the coding of residual data.[5] Two sets of context models are specified for the syntax elements significant_coeff_flag and last_significant_coeff_flag, where the coding of both syntax elements is conditioned on the scanning position as further described in Section III-B. Since in macroblock adaptive frame/field coded frames, the scanning pattern depends on the mode decision of a frame/field coded macroblock, separate sets of models have been defined for both modes. The range values in the lower row of the corresponding syntax elements in Table II specify the context indices for field-based coding mode. Note that in pure frame or field coded pictures only 277 out of the total number of 399 probability models are actually used.

The context index for coded_block_pattern is specified by the relation of (1), whereas for all other syntax elements of residual data, a context index γ is given by

$$\gamma = \Gamma_S + \Delta_S(ctx\_cat) + \chi_S \tag{2}$$

where in addition to the context index offset Γ_S the *context category* (ctx_cat) dependent offset Δ_S is employed. The specification of the context categories and the related values of Δ_S are given in Tables III and IV. By using the notation of (2), a more detailed description of the context models for syntax elements of residual data will be given in Section III-B.

Note that for the context-modeling process only past coded values of syntax elements are evaluated that belong to the same slice, where the current coding process takes place. It is also worth noting that regardless of the type of context model, conditioning is always confined to syntax element values such that the entropy encoding/decoding process can be completely decoupled from the rest of the encoding/decoding operations in a H.264/AVC encoder/decoder.

[5]As an exception, context index $\gamma = 276$ is related to the end of slice flag.

TABLE IV
BASIC BLOCK TYPES WITH NUMBER OF COEFFICIENTS AND ASSOCIATED CONTEXT CATEGORIES

BlockType	MaxNumCoeff	Context category (*ctx_cat*)
Luma DC block for Intra16x16	16	Luma-Intra16-DC: 0
Luma AC block for Intra16x16	15	Luma-Intra16-AC: 1
Luma block for Intra 4x4	16	Luma-4x4: 2
Luma block for Inter	16	
U-Chroma DC block for Intra	4	Chroma-DC: 3
V-Chroma DC block for Intra	4	
U-Chroma DC block for Inter	4	
V-Chroma DC block for Inter	4	
U-Chroma AC block for Intra	15	Chroma-AC: 4
V-Chroma AC block for Intra	15	
U-Chroma AC block for Inter	15	
V-Chroma AC block for Inter	15	

C. Binary Arithmetic Coding

Binary arithmetic coding is based on the principle of recursive interval subdivision that involves the following elementary multiplication operation. Suppose that an estimate of the probability $p_{\text{LPS}} \in (0, 0.5]$ of the *least probable symbol* (LPS) is given and that the given interval is represented by its lower bound L and its width (range) R. Based on that settings, the given interval is subdivided into two subintervals: one interval of width

$$R_{\text{LPS}} = R \cdot p_{\text{LPS}} \tag{3}$$

which is associated with the LPS, and the dual interval of width $R_{\text{MPS}} = R - R_{\text{LPS}}$, which is assigned to the most probable symbol (MPS) having a probability estimate of $1 - p_{\text{LPS}}$. Depending on the observed binary decision, either identified as the LPS or the MPS, the corresponding subinterval is then chosen as the new current interval. A binary value pointing into that interval represents the sequence of binary decisions processed so far, whereas the range of that interval corresponds to the product of the probabilities of those binary symbols. Thus, to unambiguously identify that interval and hence the coded sequence of binary decisions, the Shannon lower bound on the entropy of the sequence is asymptotically approximated by using the minimum precision of bits specifying the lower bound of the final interval.

In a practical implementation of binary arithmetic coding, the main bottleneck in terms of throughput is the multiplication operation in (3) required to perform the interval subdivision. A significant amount of work has been published in literature aimed at speeding up the required calculation in (3) by introducing some approximations of either the range R or of the probability p_{LPS} such that the multiplication can be avoided [32]–[34]. Among these low-complexity binary arithmetic coding methods, the Q coder [32] and its derivatives QM and MQ coder [35] have attracted great attention, especially in the context of the still image coding standardization groups JPEG and JBIG.

Although the MQ coder represents the state-of-the-art in fast binary arithmetic coding, we found that it considerably degrades coding efficiency, at least in the application scenario of H.264/AVC video coding [36]. Motivated by this observation, we have designed an alternative multiplication-free binary arithmetic coding scheme, the so-called *modulo coder* (M coder), which can be shown to have negligible performance degradation in comparison to a conventional binary arithmetic coder as, e.g., proposed in [37]. At the same time, our novel design of the M coder has been shown to provide a higher throughput rate than the MQ coder when compared in a software-based implementation [36].

The basic idea of our multiplication-free approach to binary arithmetic coding is to project both the legal range $[R_{\min}, R_{\max})$ of interval width R and the probability range associated with the LPS onto a small set of representative values $\boldsymbol{Q} = \{Q_0, \ldots, Q_{K-1}\}$ and $\boldsymbol{P} = \{p_0, \ldots, p_{N-1}\}$, respectively. By doing so, the multiplication on the right-hand side of (3) can be approximated by using a table of $K \times N$ pre-computed product values $Q_\rho \cdot p_\sigma$ for $\{0 \leq \rho \leq K-1\}$ and $\{0 \leq \sigma \leq N-1\}$. For the regular arithmetic core engine in H.264/AVC, a good tradeoff between a reasonable size of the corresponding table and a sufficiently good approximation of the "exact" interval subdivision of (3) was found by using a set $\boldsymbol{Q}$ of $K = 4$ quantized range values together with a set $\boldsymbol{P}$ of $N = 64$ LPS related probability values, as described in more details in Sections III-C and III-D.

Another distinct feature of the binary arithmetic coding engine in H.264/AVC, as already mentioned above, is its simplified bypass coding mode. This mode is established for specific syntax elements or parts thereof, which are assumed to be nearly uniformly distributed. For coding this kind of symbol in bypass mode, the computationally expensive probability estimation will be completely omitted.

III. Detailed Description of CABAC

This section provides detailed information for each syntax element regarding the specific choice of a binarization scheme and the associated context models for each bin of the corresponding bin string. For that purpose, the syntax elements are divided into two categories. The first category, which is described in Section III-A, contains the elements related to macroblock type, submacroblock type, and information of prediction modes both of spatial and of temporal type as well as slice- and macroblock-based control information. In the second category, which is described in Section III-B, all residual data elements, i.e., all syntax elements related to the coding of transform coefficients are combined.

In addition, a more detailed explanation of the probability estimation process and the table-based binary arithmetic coding engine of CABAC is given in Sections III-C and III-D, respectively.

A. Coding of Macroblock Type, Prediction Mode, and Control Information

1) Coding of Macroblock and Submacroblock Type: At the top level of the macroblock layer syntax the signaling of mb_skip_flag and mb_type is performed. The binary-valued mb_skip_flag indicates whether the current macroblock in a P/SP or B slice is skipped, and if it is not (i.e., mb_skip_flag $= 0$) further signaling of mb_type specifies the chosen macroblock type. For each 8×8 submacroblock of a macroblock coded in "P_8×8" or "B_8×8" mode, an additional syntax element (sub_mb_type) is present that specifies the type of the corresponding submacroblock. In this section, we will restrict our presentation to the coding of mb_type, mb_skip_flag, and sub_mb_type in P/SP slices only; for more information, the reader is referred to [1].

Macroblock Skip Flag: For coding of the mb_skip_flag, statistical dependencies between neighboring values of the syntax element mb_skip_flag are exploited by means of a simple but effective context design. For a given macroblock C, the related context models involve the mb_skip_flag values of the neighboring macroblocks to the left (denoted by A) and on top of C (denoted by B). More specifically, the corresponding context index increment χ_{MbSkip} is defined by

$$\chi_{\text{MbSkip}}(C) = (\text{mb\_skip\_flag}(A) \,!= 0) \,?\, 0 : 1 \\ + (\text{mb\_skip\_flag}(B) \,!= 0) \,?\, 0 : 1. \quad (4)$$

If one or both of the neighboring macroblocks X (A or B) are not available (e.g., because they are outside of the current slice), the corresponding mb_skip_flag (X) value in (4) is set to 0.[6]

Macroblock Type: As already stated above, for the binarization of mb_type and sub_mb_type specifically designed binarization schemes are used. Fig. 2(a) and (b), respectively, shows the corresponding binarization trees for mb_type and sub_mb_type that are used in P or SP slices, where the terminal nodes of the trees correspond to the symbol values of the syntax element and the concatenation of the binary decisions on the way from the root node to the corresponding terminal node represents the bin string of the corresponding symbol value. Note that the mb_type value of "4" for P slices is not used in CABAC entropy coding mode. For the values "5"–"30" of mb_type, which represent the intra macroblock types in a P slice, the corresponding bin strings consist of a concatenation of the prefix bin string "1", as shown in Fig. 2, and a suffix bin string, which is further specified in [1].

For coding a bin value corresponding to the binary decision at an internal node as shown in Fig. 2, separate context models denoted by $C0, \ldots, C3$ for mb_type and $C'0$, $C'1$, $C'2$ for sub_mb_type are employed.

2) Coding of Prediction Modes: Since all samples of a macroblock are predicted, the corresponding prediction modes have to be transmitted. For a macroblock coded in intra mode, these syntax elements are given by the intra prediction modes for both luminance and chrominance, whereas for an inter coded macroblock the reference picture index/indices together with their related motion vector component differences have to be signaled.

Intra Prediction Modes for Luma 4×4: The luminance intra prediction modes for 4×4 blocks are itself predicted resulting in the syntax elements of the binary-valued prev_intra4×4_pred_mode_flag and the mode indicator rem_intra4×4_pred_mode, where the latter is only present if the former takes a value of 0. For coding these syntax elements,

[6]In the following, the information about the "exception handling" in the definition of the context index increments will be (mostly) neglected. For filling that information gap, the interested reader is referred to [1].

two separate probability models are utilized: one for coding of the flag and another for coding each bin value of the 3-bit FL binarized value of rem_intra4×4_pred_mode.

Intra Prediction Modes for Chroma: Spatially neighboring intra prediction modes for the chrominance typically exhibits some correlation, which are exploited by a simple context design that relies on the related modes of the neighboring macroblocks A to the left and B on top of the current macroblock. However, not the modes of the neighbors itself are utilized for specifying the context model, but rather the binary-valued information *ChPredInDcMode*, which signals whether the corresponding mode takes the typically observed most probable mode given by the value "0" (DC prediction). Thus, the corresponding context index increment $\chi_{\mathrm{ChPred}}(C)$ for a given MB C is defined by

$$\chi_{\mathrm{ChPred}}(C) = (\mathrm{ChPredInDcMode}(A)\,!=0)\,?\,0:1 + (\mathrm{ChPredInDcMode}(B)\,!=0)\,?\,0:1. \quad (5)$$

This context design results in three models, which are only applied to the coding of the value representing the first bin of the TU binarization of the current value of the syntax element intra_chroma_pred_mode to encode. Since the value of the first bin carries the ChPredInDcMode (C) information of the current macroblock C, it is reasonable to restrict the application of the three models defined by (5) to this bin only. For the two remaining bins, an additional (fixed) probability model is applied.

Reference Picture Index: The same design principle as before was applied to the construction of the context models for the reference picture index ref_idx.[7] First, the relevant information for conditioning the first bin value of ref_idx is extracted from the reference picture indices of the neighboring macroblock or submacroblock partitions A to the left and B on top of the current partition C. This information is appropriately condensed in the binary flag *RefIdxZeroFlag*, which indicates whether ref_idx with value 0 is chosen for the corresponding partition. As a slight variation, the related context index increment was chosen to represent four instead of three models as in the previously discussed context designs

$$\chi_{\mathrm{RefIdx}}(C) = (\mathrm{RefIdxZeroFlag}(A)\,!=0)\,?\,0:1 + 2\cdot((\mathrm{RefIdxZeroFlag}(B)\,!=0)\,?\,0:1).$$

Application of these context models to the first bin of the unary binarized reference picture index is complemented by the usage of two additional probability models for encoding of the values related to the second and all remaining bins.

Components of Motion Vector Differences: Motion vector differences are prediction residuals, for which a context model is established in CABAC that is based on the local prediction error. Let $mvd(X, cmp)$ denote the value of a motion vector difference component of direction $cmp \in \{\mathrm{horizontal}, \mathrm{vertical}\}$ related to a macroblock or submacroblock partition X. Then, the related context index increment $\chi_{\mathrm{Mvd}}(C, cmp)$ for a given macroblock or submacroblock partition C and component (cmp) is determined by

$$\chi_{\mathrm{Mvd}}(C, cmp) = \begin{cases} 0, & \text{if } e(A,B,cmp) < 3 \\ 1, & \text{if } 3 \le e(A,B,cmp) \le 32 \\ 2, & \text{if } e(A,B,cmp) > 32 \end{cases}$$
$$\text{with } e(A,B,cmp) = |mvd(A,cmp)| + |mvd(B,cmp)| \quad (6)$$

where A and B represent the corresponding macroblock or submacroblock partitions to the left and on the top of the regarded macroblock or submacroblock partition C, respectively.[8] $\chi_{\mathrm{Mvd}}(C, cmp)$ in (6) is only applied for the selection of the probability model that is used to code the value of the first bin of the binarized value of $mvd(C, cmp)$. Binarization of motion vector differences is achieved by applying the UEG3 binarization scheme with a cutoff value of 9. That implies, in particular, that only the unary prefix part is encoded in regular coding mode, where four additional context models are employed for coding the values related to the second, third, fourth, and fifth to ninth bin of the prefix part. The values of the bins related to the Exp-Golomb suffix part including the sign bit are encoded using the bypass coding mode.

3) Coding of Control Information: Three additional syntax elements are signaled at the macroblock or macroblock pair level, which we refer to as control information. These elements are given by mb_qp_delta, end_of_slice_flag, and mb_field_decoding_flag.

Macroblock-based Quantization Parameter Change: For updating the quantization parameter on a macroblock level, mb_qp_delta is present for each nonskipped macroblock with a value of coded_block_pattern unequal to 0.[9] For coding the signed value $\delta(C)$ of this syntax element for a given macroblock C in CABAC, $\delta(C)$ is first mapped onto a positive value $\delta^+(C)$ by using the relation

$$\delta^+(C) = 2|\delta(C)| - ((\delta(C) > 0)\,?\,1:0).$$

Then, $\delta^+(C)$ is binarized using the unary binarization scheme. For encoding the value of the corresponding first bin, a context model is selected based on the binary decision $(\delta(P)\,!=0)$ for the preceding macroblock P of C in decoding order. This results in two probability models for the first bin, whereas for the second and all remaining bins, two additional probability models are utilized.

End of Slice Flag: For signaling the last macroblock (or macroblock pair) in a slice, the end_of_slice_flag is present for each macroblock (pair). It is encoded using a specifically designed nonadaptive probability model such that the event of a nonterminating macroblock (pair) is related to the highest possible MPS probability (see Section III-C for more details on the related probability model).

Macroblock Pair Field Flag: In macroblock adaptive frame/field coded frames, the mb_field_decoding_flag signals for each macroblock pair whether it is coded in frame or

[7]For clarity of presentation, the reference picture list suffices l0 and l1 are suppressed in the following exposition, both for the reference picture index and the motion vector difference.

[8]The precise definition of a neighboring partition used for context modeling of both the reference picture index and the motion vector difference is given in [1].

[9]For macroblocks coded in an "intra_16×16" prediction mode, the syntax element mb_qp_delta is always present.

field coding mode. For coding this flag in CABAC, spatial correlations between the coding mode decisions of neighboring macroblock pairs are exploited by choosing between three probability models. For a given macroblock pair C, the selection of the corresponding model is performed by means of the related context index increment $\chi_{\mathrm{MbField}}(C)$, which is defined as

$$\chi_{\mathrm{MbField}}(C) = \mathrm{mb\_field\_decoding\_flag}(A) + \mathrm{mb\_field\_decoding\_flag}(B).$$

Here, A and B represent the corresponding macroblock pairs to the left and on the top of the current macroblock pair C.

B. Coding of Residual Data

1) Characteristic Features: For the coding of residual data within the H.264/AVC standard specifically designed syntax elements are used in CABAC entropy coding mode. These elements and their related coding scheme are characterized by the following distinct features.

- A one-bit symbol coded_block_flag and a binary-valued significance map are used to indicate the occurrence and the location of nonzero transform coefficients in a given block.
- Non-zero levels are encoded in reverse scanning order.
- Context models for coding of nonzero transform coefficients are chosen based on the number of previously transmitted nonzero levels within the reverse scanning path.

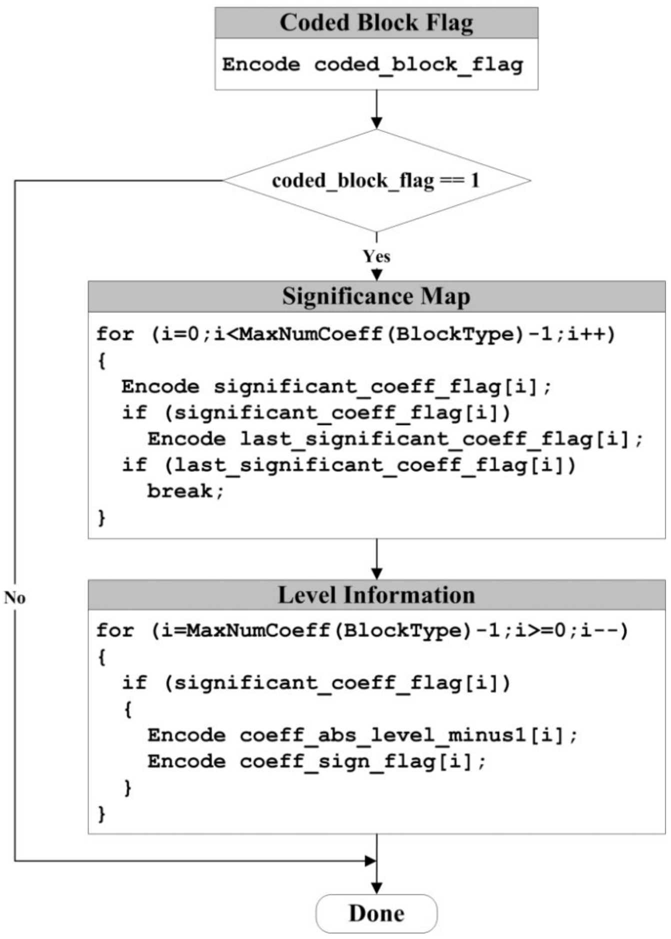
2) Encoding Process for Residual Data: Fig. 5 illustrates the CABAC encoding scheme for a single block of transform coefficients.

First, the coded block flag is transmitted for the given block of transform coefficients unless the coded block pattern or the macroblock mode indicates that the regarded block has no nonzero coefficients. If the coded block flag is zero, no further information is transmitted for the block; otherwise, a significance map specifying the positions of significant coefficients is encoded. Finally, the absolute value of the level as well as the sign is encoded for each significant transform coefficient. These values are transmitted in reverse scanning order.

In the following, a more detailed description of each of the major building blocks of Fig. 5 is given together with a brief specification of the CABAC encoding procedure for the coded_block_pattern symbol.

Coded Block Pattern: For each nonskipped macroblock with prediction mode not equal to intra_16×16, the coded_block_pattern symbol indicates which of the six 8×8 blocks—four for luminance and two for chrominance—contain nonzero transform coefficients. A given value of the syntax element coded_block_pattern is binarized using the concatenation of a 4-bit FL and a TU binarization with cutoff value $S = 2$, as already noted in Section II-A.

Coded Block Flag: coded_block_flag is a one-bit symbol, which indicates if there are significant, i.e., nonzero coefficients inside a single block of transform coefficients, for which the coded block pattern indicates nonzero entries. If coded_block_flag is zero, no further information is transmitted for the related block.



```
Encode coded_block_flag
```

```
for (i=0;i<MaxNumCoeff(BlockType)-1;i++)
{
  Encode significant_coeff_flag[i];
  if (significant_coeff_flag[i])
    Encode last_significant_coeff_flag[i];
  if (last_significant_coeff_flag[i])
    break;
}
```

```
for (i=MaxNumCoeff(BlockType)-1;i>=0;i--)
{
  if (significant_coeff_flag[i])
  {
    Encode coeff_abs_level_minus1[i];
    Encode coeff_sign_flag[i];
  }
}
```

Fig. 5. Flow diagram of the CABAC encoding scheme for a block of transform coefficients.

Scanning of Transform Coefficients: The 2-D arrays of transform coefficient levels of those subblocks for which the coded_block_flag indicates nonzero entries are first mapped onto a one-dimensional list using a given scanning pattern.

Significance Map: If the coded_block_flag indicates that a block has significant coefficients, a binary-valued significance map is encoded. For each coefficient in scanning order, a one-bit symbol significant_coeff_flag is transmitted. If the significant_coeff_flag symbol is one, i.e., if a nonzero coefficient exists at this scanning position, a further one-bit symbol last_significant_coeff_flag is sent. This symbol indicates if the current significant coefficient is the last one inside the block or if further significant coefficients follow. Table V shows an example for the significance map encoding procedure.

Note that the flags (significant_coeff_flag, last_significant_coeff_flag) for the last scanning position of a block are never transmitted. If the last scanning position is reached and the significance map encoding was not already terminated by a last_significant_coeff_flag with value one, it is obvious that the last coefficient has to be significant.

Level Information: The encoded significance map determines the locations of all significant coefficients inside a block of quantized transform coefficients. The values of the significant coefficients (levels) are encoded by using two coding symbols: coeff_abs_level_minus1 (representing the absolute value of

TABLE V
EXAMPLE FOR ENCODING THE SIGNIFICANCE MAP

Scanning position	1	2	3	4	5	6	7	8	9
Transform coeff. levels	9	0	-5	3	0	0	-1	0	1
significant_coeff_flag	1	0	1	1	0	0	1	0	1
last_significant_coeff_flag	0		0	0			0		1

the level minus 1), and coeff_sign_flag (representing the sign of levels). While coeff_sign_flag is a one-bit symbol (with values of 1 for negative coefficients), the UEG0 binarization scheme, as depicted in Table I is used for encoding the values of coeff_abs_level_minus1. The levels are transmitted in reverse scanning order (beginning with the last significant coefficient of the block) allowing the usage of reasonably adjusted context models, as described in the next paragraph.

3) Context Models for Residual Data: In H.264/AVC residual data coding, there are 12 different types of transform coefficient blocks (denoted by BlockType in left column of Table IV), which typically have different kinds of statistics.

However, for most sequences and coding conditions some of the statistics are very similar. To keep the number of different context models used for coefficient coding reasonably small, the block types are classified into five categories as specified in the right column of Table IV. For each of these categories, a special set of context models is used for all syntax elements related to residual data with the exception of coded_block_pattern, as further described in the next paragraph.

Coded Block Pattern: Since each of the bits in the bin string of coded_block_pattern represents a coding decision of a corresponding block of transform coefficients, the chosen probability models for that syntax element depend on the bin index. For bin indices from 0 to 3 corresponding to the four 8×8 luminance blocks, the context index increment χ_{CBP} for a given 8×8 block C related to bin index bin_idx is given by

$$\chi_{\mathrm{CBP}}(C, \mathrm{bin\_idx}) = ((\mathrm{CBP\_Bit}(A)\,!=0)\,?\,0:1) \\ +2\cdot((\mathrm{CBP\_Bit}(B)\,!0)\,?\,0:1)$$

where $\mathrm{CBP\_Bit}(A)$ and $\mathrm{CBP\_Bit}(B)$ represent the bit of the coded block pattern corresponding to the 8×8 blocks A to the left and B on the top of the regarded block C, respectively.

For each of the bin indices 4 and 5, which are related to the two "chrominance" bins in the binarized value of coded_block_pattern, a similar context assignment rule as for the luminance bins is defined such that for both bin indices together eight additional probability models are specified [1].

Coded Block Flag: Coding of the coded_block_flag utilizes four different probability models for each of the five categories as specified in Table IV. The context index increment $\chi_{\mathrm{CBFlag}}(C)$ for a given block C is determined by

$$\chi_{\mathrm{CBFlag}}(C) = \mathrm{coded\_block\_flag}(A) \\ +2\cdot\mathrm{coded\_block\_flag}(B) \quad (7)$$

where A and B represent the corresponding blocks of the same type to the left and on the top of the regarded block C, respectively. Only blocks of the same type are used for context determination. The following block types are differentiated: Luma-DC, Luma-AC, Chroma-U-DC, Chroma-U-AC, Chroma-V-DC, and Chroma-V-AC. If no neighboring block X (A or B) of the same type exists (e.g., because the current block is intra_16×16 coded and the neighboring block X is inter coded), the corresponding $\mathrm{coded\_block\_flag}(X)$ value in (7) is set to 0. If a neighboring block X (A or B) is outside the picture area or positioned in a different slice, the corresponding $\mathrm{coded\_block\_flag}(X)$ value is replaced by a default value. If the current block C is coded using an intra prediction mode, a default value of one is used; otherwise, a default value of zero is used. Thus, while six block types are distinguished for determining the context increment, five different sets of models (each for one category specified in the right column of Table IV) are used for encoding the coded block flag. This results in a total number of 20 different probability models for the coded_block_flag bit.

Significance Map: For encoding the significance map, up to 15 different probability models are used for both the significant_coeff_flag and the last_significant_coeff_flag. The choice of the models and thus the corresponding context index increments χ_{SIG} and χ_{LAST} depend on the scanning position, i.e., for a coefficient $\mathrm{coeff}[i]$, which is scanned at the ith position, the context index increments are determined as follows:

$$\chi_{\mathrm{SIG}}(\mathrm{coeff}[i]) = \chi_{\mathrm{LAST}}(\mathrm{coeff}[i]) = i.$$

Depending on the maximum number of coefficients (MaxNumCoeff) for each context category as given in Table IV, this results in MaxNumCoeff-1 different contexts. Thus, a total number of 61 different models for both the significant_coeff_flag and the last_significant_coeff_flag is reserved.

Level Information: Reverse scanning of the level information allows a more reliable estimation of the statistics, because at the end of the scanning path it is very likely to observe the occurrence of successive so-called trailing 1's, i.e., transform coefficient levels with absolute value equal to 1. Consequently, for encoding coeff_abs_level_minus1, two adequately designed sets of context models are used: one for the first bin (with bin index 0) and another one for the remaining bins of the UEG0 prefix bin string with indices 1 to 13.

Let $\mathrm{NumT1}(i)$ denote the accumulated number of already encoded/decoded trailing 1's, and let $\mathrm{NumLgt1}(i)$ denote the accumulated number of encoded/decoded levels with absolute value greater than 1, where both counters are related to the current scanning position i within the processed transform coefficient block. Note that both counters are initialized with the value of 0 at the beginning of the reverse scanning of levels and that the numbers of both counters are monotonically increasing with decreasing scanning index i along the backward scanning path. Then, the context for the first bin of coeff_abs_level_minus1 is determined by the current value NumT1, where the following additional rules apply. If more than three past coded coefficients have an absolute value of 1, the context index increment of three is always chosen. When a level with an absolute value greater than 1 has been encoded, i.e., when NumLgt1 is greater than 0, a context index increment of 4 is used for all remaining levels of the regarded block. Thus, for encoding the first bin

TABLE VI
EXAMPLE FOR DETERMINATION OF CONTEXT INDEX INCREMENTS FOR ENCODING THE ABSOLUTE VALUES OF TRANSFORM COEFFICIENT LEVELS

Scanning position i	1	2	3	4	5	6	7	8	9
Transform coeff. levels	9	0	-5	3	0	0	-1	0	1
$\chi_{\text{AbsCoeff}}(i$, bin_index = 0)	4		4	2			1		0
$\chi_{\text{AbsCoeff}}(i$, bin_index > 0)	7		6	5					

with bin_index $= 0$, as shown in the second (light gray-shaded) column of Table I, the corresponding context index increment $\chi_{\text{AbsCoeff}}(i, \text{bin\_index} = 0)$ at the scanning position i is given as

$$\chi_{\text{AbsCoeff}}(i, \text{bin\_index} = 0) = \begin{cases} 4, & \text{if } \text{NumLgt1}(i) > 0 \\ \max(3, \text{NumT1}(i)), & \text{otherwise.} \end{cases}$$

For encoding the bins with $1 \leq \text{bin\_index} \leq 13$ (as shown in the dark gray-shaded columns in Table I), the context index increment $\chi_{\text{AbsCoeff}}(i, \text{bin\_index})$ is determined by NumLgt1 with a maximum context index increment of 4, i.e.,

$$\chi_{\text{AbsCoeff}}(i, \text{bin\_index}) = 5 + \max(4, \text{NumLgt1}(i)) \quad (1 \leq \text{bin\_index} \leq 13).$$

For all bins of the UEG0 suffix part of coeff_abs_level_minus1 (with bin index greater than 13) as well as for the sign information coeff_sign_flag, the bypass coding mode is used for all block types. Thus, the total number of different probability models for encoding the level information is 49.[10]

Table VI shows an example of the determination of the context index increment $\chi_{\text{AbsCoeff}}(i, \text{bin\_index})$ used for encoding the absolute value of levels of significant transform coefficients. Note that the transform coefficient levels are processed in reverse scanning order, i.e., from the ninth position to the first position in scanning order.

C. *Probability Estimation*

As outlined in Section II-C, the basic idea of the new multiplication-free binary arithmetic coding scheme for H.264/AVC relies on the assumption that the estimated probabilities of each context model can be represented by a sufficiently limited set of representative values. For CABAC, 64 representative probability values $p_\sigma \in [0.018\,75, 0.5]$ were derived for the LPS by the following recursive equation:

$$p_\sigma = \alpha \cdot p_{\sigma-1} \text{ for all } \sigma = 1, \ldots, 63$$
$$\text{with } \alpha = \left(\frac{0.018\,75}{0.5}\right)^{1/63} \text{ and } p_0 = 0.5. \qquad (8)$$

Here, both the chosen scaling factor $\alpha \approx 0.95$ and the cardinality $N = 64$ of the set of probabilities represent a good compromise between the desire for fast adaptation ($\alpha \rightarrow 0$; small N), on one hand, and the need for a sufficiently stable and accurate estimate ($\alpha \rightarrow 1$; larger N), on the other hand. Note that unlike, e.g., in the MQ coder, there is no need to tabulate the representative LPS probability values $\{p_\sigma \mid 0 \leq \sigma \leq 63\}$ in the CABAC approach. As further described below, each probability value p_σ is only implicitly addressed in the arithmetic coding engine by its corresponding index σ.

[10]Note that for the chrominance DC blocks, there are only four different models for the bins of coeff_abs_level_minus1 with indices 1 to 13, since at maximum four nonzero levels are transmitted.

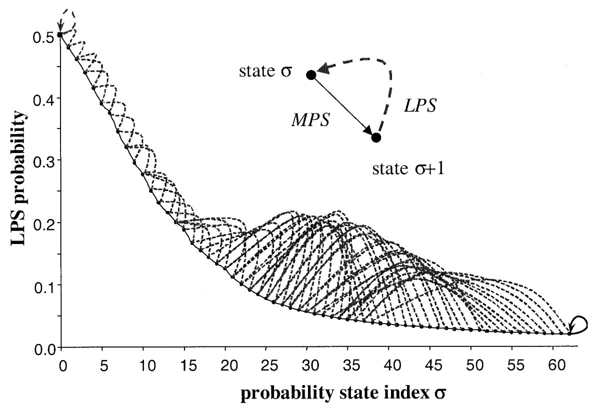


Fig. 6. LPS probability values and transition rules for updating the probability estimation of each state after observing a LPS (dashed lines in left direction) and a MPS (solid lines in right direction).

As a result of this design, each context model in CABAC can be completely determined by two parameters: its current estimate of the LPS probability, which in turn is characterized by an index σ between 0 and 63, and its value of MPS ϖ being either 0 or 1. Thus, probability estimation in CABAC is performed by using a total number of 128 different probability states, each of them efficiently represented by a 7-bit integer value. In fact, one of the state indices ($\sigma = 63$) is related to an autonomous, nonadaptive state with a fixed value of MPS, which is only used for encoding of binary decisions before termination of the arithmetic codeword, as further explained below. Therefore, only 126 probability states are effectively used for the representation and adaptation of all (adaptive) context models.

1) Update of Probability States: As already stated above, all probability models in CABAC with one exception are (backward) adaptive models, where an update of the probability estimation is performed after each symbol has been encoded. Actually, for a given probability state, the update depends on the state index and the value of the encoded symbol identified either as a LPS or a MPS. As a result of the updating process, a new probability state is derived, which consists of a potentially modified LPS probability estimate and, if necessary, a modified MPS value.

Fig. 6 illustrates the probability values $\{p_\sigma \mid 0 \leq \sigma \leq 62\}$ for the LPS estimates together with their corresponding transition rules for updating the state indices. In the event of a MPS, a given state index is simply incremented by 1, unless a MPS occurs at state index 62, where the LPS probability is already at its minimum, or equivalently, the maximum MPS probability is reached. In the latter case, the state index 62 remains fixed until a LPS is seen, in which case the state index is changed by decrementing the state index by an amount illustrated by the dashed line in Fig. 6. This rule applies in general to each occurrence of a LPS with the following exception. Assuming a LPS has been encoded at the state with index $\sigma = 0$, which corresponds to the equi-probable case, the state index remains fixed, but the MPS value ϖ will be toggled such that the value of the LPS and MPS will be interchanged. In all other cases, no matter which symbol has been encoded, the MPS value will not be altered. The derivation of the transition rules for the LPS probability is based on the following relation between a given LPS probability p_{old} and its updated counterpart p_{new}:

$$p_{\text{new}} = \begin{cases} \max(\alpha \cdot p_{\text{old}}, p_{62}), & \text{if a MPS occurs} \\ \alpha \cdot p_{\text{old}} + (1-\alpha), & \text{if a LPS occurs} \end{cases}$$

where the value of α is given as in (8).

With regard to a practical implementation of the probability estimation process in CABAC, it is important to note that all transition rules can be realized by at most two tables each having 63 entries of 6-bit unsigned integer values. Actually, it is sufficient to provide a single table TransIdxLPS, which determines for a given state index σ the new updated state index $\text{TransIdxLPS}[\sigma]$ in case an LPS has been observed.[11] The MPS-driven transitions can be obtained by a simple (saturated) increment of the state index σ by the fixed value of 1 resulting in an updated state index $\min(\sigma + 1, 62)$.

2) Initialization and Reset of Probability States: The basic self-contained unit in H.264/AVC video coding is a slice. This fact implies in particular certain restrictions on the backward adaptation process of probability models as described in the previous paragraph. Since the lifetime of the backward adaptation cannot exceed the duration of the whole slice encoding process, which in turn may represent a substantial amount of the whole adaptation process, all models have to be re-initialized at the slice boundaries using some pre-defined probability states. In the absence of any prior knowledge about the source, one possible choice would be to initialize each model with the equi-probable state. However, CABAC provides a built-in mechanism for incorporating some *a priori* knowledge about the source statistics in the form of appropriate initialization values for each of the probability models. This so-called *initialization process for context models* allows an adjustment of the initial probability states in CABAC on two levels.

Quantization Parameter Dependent Initialization: On the lower level of adjustment, there is a default set of initialization values, which are derived from the initially given slice quantization parameter SliceQP, thus providing some kind of pre-adaptation of the initial probability states to the different coding conditions represented by the current value of the SliceQP parameter.

Training sequences have been used to fit the initial probability state of each model to the quantization parameter. By using a linear regression, a pair of parameters (μ_γ, ν_γ) was obtained for each probability model with context index $\gamma (0 \leq \gamma \leq 398, \gamma \neq 276)$, from which the corresponding SliceQP dependent initial probability state is derived during the initialization process by applying the procedure shown in Fig. 7.

```
1. $\sigma_{\text{pre}}$ =
       max(1, min(126,(( $\mu_\gamma$ * SliceQP ) >> 4) + $\nu_\gamma$))
2. if( $\sigma_{\text{pre}}$ <= 63 ) {
     $\sigma$ = 63 - $\sigma_{\text{pre}}$
     $\varpi$ = 0
   } else {
     $\sigma$ = $\sigma_{\text{pre}}$ - 64
     $\varpi$ = 1
   }
```

Fig. 7. SliceQP dependent initialization procedure. First, one of the admissible probability states σ_{pre} (numbered between 1 and 126) is derived from the given parameters (μ_γ, ν_γ) and SliceQP. Then, in a second step σ_{pre} is translated into the probability state index σ and the value of the MPS (ϖ).

Slice-Dependent Initialization: The concept of a low-level pre-adaptation of the probability models was generalized by defining two additional sets of context initialization parameters for those probability models specifically used in P and B slices. In this way, the encoder is enabled to choose for these slice types between three initialization tables such that a better fit to different coding scenarios and/or different types of video content can be achieved.[12] This forward-adaptation process requires the signaling of the chosen initialization table, which is done by specifying the corresponding table index (0–2) in the slice header. In addition, an increased amount of memory for the storage of the initialization tables is required. However, access to this memory of approximately 3 kB is needed only once per slice. Note that a chosen initialization table triggers for each probability model the same low-level SliceQP dependent initialization procedure as described in the previous paragraph. Depending on the slice size and the bit rate which, in turn, depend on the amount of data that can be used for the backward adaptation process of the symbol statistics, bit-rate savings of up to 3% have been obtained by using the instrument of slice-dependent context initialization [13].

D. Table-Based Binary Arithmetic Coding

In this section, we present some more detailed information about the binary arithmetic coding engine of H.264/AVC. Actually, the CABAC coding engine consists of two subengines, one for the regular coding mode, which includes the utilization of adaptive probability models, and another so-called "bypass" coding engine for a fast encoding of symbols, for which an approximately uniform probability is assumed to be given. The following presentation of the basic features of the CABAC coding engine also involves aspects of renormalization, carry-over control, and termination.

1) Interval Subdivision in Regular Coding Mode: Fig. 8 illustrates the binary arithmetic encoding process for a given bin value *binVal* using the regular coding mode. The internal state of the arithmetic encoding engine is as usual characterized by two quantities: the current interval range R and the base (lower endpoint) L of the current code interval. Note, however, that the precision needed to store these registers in the CABAC engine (both in regular and bypass mode) can be reduced up to 9 and 10 bits, respectively. Encoding of the given binary value *binVal* observed in a context with probability state index σ and value

[11]The specific values of this state transition table can be found in [1].

[12]For a description of an example of the nonnormative table selection process, please refer to [13].

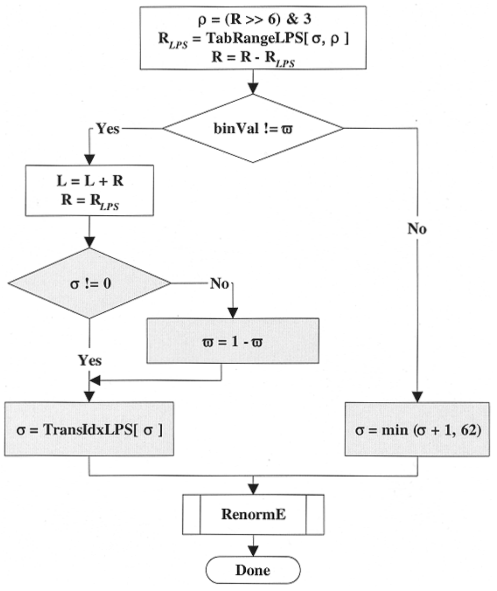


Fig. 8. Flow diagram of the binary arithmetic encoding process including the updating process of probability estimation (in gray shaded boxes) for a single bin value (*binVal*) using the regular coding mode.

of $\mathrm{MPS}\,\varpi$ is performed in a sequence of four elementary steps as follows.

In the first and major step, the current interval is subdivided according to the given probability estimates. This interval subdivision process involves three elementary operations as shown in the topmost box of the flow diagram in Fig. 8. First, the current interval range R is approximated by a quantized value $Q(R)$ using an equi-partition of the whole range $2^8 \leq R < 2^9$ into four cells. But instead of using the corresponding representative quantized range values Q_0, Q_1, Q_2, and Q_3 explicitly in the CABAC engine, $Q(R)$ is only addressed by its quantizer index ρ, which can be efficiently computed by a combination of a shift and bit-masking operation, i.e.,

$$\rho = (R >> 6)\,\&\,3.$$

Then, this index ρ and the probability state index σ are used as entries in a 2-D table TabRangeLPS to determine the (approximate) LPS related subinterval range R_{LPS}, as shown in Fig. 8. Here, the table TabRangeLPS contains all 64×4 pre-computed product values $p_\sigma \cdot Q_\rho$ for $0 \leq \sigma \leq 63$ and $0 \leq \rho \leq 3$ in 8-bit precision.[13]

Given the dual subinterval range $R - R_{\mathrm{LPS}}$ for the MPS, the subinterval corresponding to the given bin value *binVal* is chosen in the second step of the encoding process. If *binVal* is equal to the MPS value ϖ, the lower subinterval is chosen so that L is unchanged (right path of the branch in Fig. 8); otherwise, the upper subinterval with range equal to R_{LPS} is selected (left branch in Fig. 8).

In the third step of the regular arithmetic encoding process the update of the probability states is performed as described in Section III-C (gray shaded boxes in Fig. 8), and finally, the fourth step consists of the renormalization of the registers L and R ("RenormE" box in Fig. 8) as further described below.

[13]For the specific values of TabRangeLPS, the reader is referred to [1].

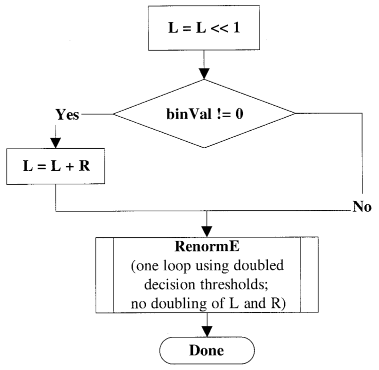


Fig. 9. Flow diagram of the binary arithmetic encoding process for a single bin value (*binVal*) using the bypass coding mode.

2) Bypass Coding Mode: To speed up the encoding (and decoding) of symbols, for which $R - R_{\mathrm{LPS}} \approx R_{\mathrm{LPS}} \approx R/2$ is assumed to hold, the regular arithmetic encoding process as described in the previous paragraph is simplified to a large extent. First, a "bypass" of the probability estimation and update process is established, and second, the interval subdivision is performed such that two equisized subintervals are provided in the interval subdivision stage. But instead of explicitly halving the current interval range R, the variable L is doubled before choosing the lower or upper subinterval depending on the value of the symbol to encode (0 or 1, respectively). In this way, doubling of L and R in the subsequent renormalization is no longer required provided that the renormalization in the bypass is operated with doubled decision thresholds (see Fig. 9).

At this point, however, one might argue that ideally no arithmetic operation would be required if the binary symbols to encode would be directly written to the bitstream, i.e., if the whole arithmetic coding engine would be bypassed. Since it is by no means a trivial task to multiplex raw bits with an arithmetic codeword without performing some kind of termination of the codeword, and since the bypass coding mode is intended to be used for symbols which, in general, are not grouped together, this kind of "lazy coding mode" as e.g., established in JPEG2000 [35] is ruled out in the present context.

3) Renormalization and Carry-Over Control: A renormalization operation after interval subdivision is required whenever the new interval range R no longer stays within its legal range of $[2^8, 2^9)$. Each time renormalization must be carried out, one or more bits can be output. However, in certain cases, the polarity of the output bits will be resolved in subsequent output steps, i.e., carry propagation might occur in the arithmetic encoder. For the CABAC engine, the renormalization process and carry-over control of [37] was adopted. This implies, in particular, that the encoder has to resolve any carry propagation by monitoring the bits that are outstanding for being emitted. More details can be found in [1].

4) Termination of Arithmetic Code Word: A special fixed, i.e., nonadapting probability state with index $\sigma = 63$ was designed such that the associated table entries $\mathrm{TabRangeLPS}[63, \rho]$, and hence R_{LPS} are determined by a fixed value of 2 regardless of the given quantized range index ρ. This guarantees that for the terminating syntax element in a slice which, in general, is given by the LPS value of the end

TABLE VII
INTERLACED TEST SEQUENCES

Name	Resolution	Frame rate	Duration
Canoe	720 × 576	25 Hz	6 sec.
Formula 1	720 × 576	25 Hz	6 sec.
Rugby	720 × 576	25 Hz	6 sec.
Mobile & Calendar	720 × 480	30 Hz	6 sec.
Football	720 × 480	30 Hz	6 sec.

of slice flag,[14] 7 bits of output are produced in the related renormalization step. Two more disambiguating bits are needed to terminate the arithmetic codeword for a given slice. By preventing the decoder from performing a renormalization after recovering the terminating syntax element, the decoder never reads more bits than were actually produced by the encoder for that given slice.

IV. Experimental Results

In our experiments for evaluating the coding efficiency of CABAC, we addressed the coding of television sequences for broadcast, a typical target application for the Main profile of H.264/AVC. The set of interlaced standard definition sequences used for testing CABAC is listed in Table VII.

All simulations were performed using the Main profile of H.264/AVC. An IDR-picture was inserted every 500 ms and two nonreference B frames were inserted between each pair of anchor frames. The motion search was conducted in a range of $[-32 \ldots 32] \times [-32 \ldots 32]$ samples for three reference frames. All encoding mode decisions including the motion search, the macroblock mode decision, and the macroblock and picture-based frame/field decision were performed using the simple and effective Lagrangian coder control presented in [38]. Bit rates were adjusted by using fixed values of the quantization parameter (QP) for an entire sequence. The value of QP for B pictures was set to $\mathrm{QP}(B) = \mathrm{QP}(I/P) + 2$, where $\mathrm{QP}(I/P)$ is the quantization parameter for I and P pictures.

In our experiments, we compare the coding efficiency of CABAC to the coding efficiency of the baseline entropy coding method of H.264/AVC. The baseline entropy coding method uses the zero-order Exp-Golomb code for all syntax elements with the exception of the residual data, which are coded using the coding method of CAVLC [1], [2].

In Fig. 10, the bit-rate savings of CABAC relative to the default entropy coding method of H.264/AVC are shown against the average PSNR of the luminance component for the five interlaced sequences of the test set. It can be seen that CABAC significantly outperforms the baseline entropy coding method of H.264/AVC for the typical area of target applications. For the range of acceptable video quality for broadcast application of about 30–38 dB and averaged over all tested sequences, bit-rate savings of 9% to 14% are achieved, where higher gains are obtained at lower rates.

[14]Another terminating symbol is given by the LPS value of the bin of the macroblock type indicating the PCM mode as further specified in [1].

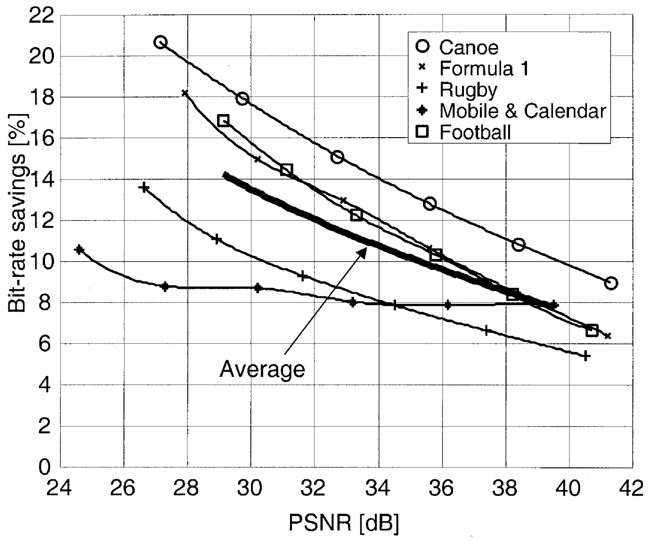


Fig. 10. Bit-rate savings provided by CABAC relative to the baseline entropy coding method CAVLC of H.264/AVC.

V. Conclusion

The CABAC design is based on the key elements of binarization, context modeling, and binary arithmetic coding. Binarization enables efficient binary arithmetic coding via a unique mapping of nonbinary syntax elements to a sequence of bits, which are called bins. Each bin can either be processed in the regular coding mode or the bypass mode. The latter is chosen for selected bins in order to allow a speedup of the whole encoding (and decoding) process by means of a simplified nonadaptive coding engine without the usage of probability estimation. The regular coding mode provides the actual coding benefit, where a bin may be context modeled and subsequently arithmetic encoded. As a design decision, in general only the most probable bin of a syntax element is context modeled using previously coded/decoded bins. Moreover, all regular coded bins are adapted by estimating their actual pdf.

The estimation of the pdf and the actual binary arithmetic coding/decoding is conducted using an efficient table-based approach. This multiplication-free method enables efficient implementations in hardware and software.

The CABAC entropy coding method is part of the Main profile of H.264/AVC [1] and may find its way into video streaming, broadcast, or storage applications within this profile. Experimental results have shown the superior performance of CABAC in comparison to the baseline entropy coding method of VLC/CAVLC. For typical test sequences in broadcast applications, averaged bit-rate savings of 9% to 14% corresponding to a range of acceptable video quality of about 30–38 dB were obtained.

Acknowledgment

The authors would like to thank G. Blättermann for implementing and testing parts of the algorithm. Thanks to G. Heising, L. Winger, F. Bossen, R. Kurceren, M. Karczewicz, J. Lainema, B. Haskell, T. Suzuki, Y. Kikuchi, R. van der Vleuten, and G. Sullivan for useful discussions.

REFERENCES

[1] "Draft ITU-T Recommendation H.264 and Draft ISO/IEC 14 496-10 AVC," in *Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-G050*, T. Wieg, Ed., Pattaya, Thailand, Mar. 2003.
[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 560–576, July 2003.
[3] "Generic Coding of Moving Pictures and Associated Audio Information—Part 2: Video," ITU-T and ISO/IEC JTC1, ITU-T Recommendation H.262—ISO/IEC 13 818-2 (MPEG-2), 1994.
[4] "Video Coding for Low Bitrate Communications, Version 1," ITU-T, ITU-T Recommendation H.263, 1995.
[5] "Coding of Audio-Visual Objects—Part 2: Visual," ISO/IEC JTC1, ISO/IEC 14 496-2 (MPEG-4 Visual version 1), Apr. 1999; Amendment 1 (version 2), Feb. 2000; Amendment 4 (streaming profile), Jan. 2001.
[6] C. A. Gonzales, "DCT Coding of Motion Sequences Including Arithmetic Coder," ISO/IEC JCT1/SC2/WP8, MPEG 89/187, MPEG 89/187, 1989.
[7] D. Marpe, G. Blättermann, and T. Wiegand, "Adaptive Codes for H.26L,", Eibsee, Germany, ITU-T SG16/Q.6 Doc. VCEG-L13, 2001.
[8] D. Marpe, G. Blättermann, G. Heising, and T. Wiegand, "Further Results for CABAC Entropy Coding Scheme,", Austin, TX, ITU-T SG16/Q.6 Doc. VCEG-M59, 2001.
[9] D. Marpe, G. Blättermann, and T. Wiegand, "Improved CABAC,", Pattaya, Thailand, ITU-T SG16/Q.6 Doc. VCEG-O18, 2001.
[10] D. Marpe, G. Blättermann, T. Wiegand, R. Kurceren, M. Karczewicz, and J. Lainema, "New results on improved CABAC," in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-B101, Geneva, Switzerland, Feb. 2002.
[11] H. Schwarz, D. Marpe, G. Blättermann, and T. Wiegand, "Improved CABAC," in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-C060, Fairfax, VA, Mar. 2002.
[12] D. Marpe, G. Heising, G. Blättermann, and T. Wiegand, "Fast arithmetic coding for CABAC," in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-C061, Fairfax, VA, Mar. 2002.
[13] H. Schwarz, D. Marpe, and T. Wiegand, "CABAC and slices," in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-D020, Klagenfurt, Austria, July 2002.
[14] M. Karczewicz, "Analysis and simplification of intra prediction," in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-D025, Klagenfurt, Austria, July 2002.
[15] D. Marpe, G. Blättermann, G. Heising, and T. Wiegand, "Proposed cleanup changes for CABAC," in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-E059, Geneva, Switzerland, Oct. 2002.
[16] F. Bossen, "CABAC cleanup and complexity reduction," in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-E086, Geneva, Switzerland, Oct. 2002.
[17] D. Marpe, H. Schwarz, G. Blättermann, and T. Wiegand, "Final CABAC cleanup," in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-F039, Awaji, Japan, Dec. 2002.
[18] D. Marpe and H. L. Cycon, "Very low bit-rate video coding using wavelet-based techniques," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 85–94, Apr. 1999.
[19] G. Heising, D. Marpe, H. L. Cycon, and A. P. Petukhov, "Wavelet-Based very low bit-rate video coding using image warping and overlapped block motion compensation," *Proc. Inst. Elect. Eng.—Vision, Image and Signal Proc.*, vol. 148, no. 2, pp. 93–101, Apr. 2001.
[20] S.-J. Choi and J. W. Woods, "Motion-compensated 3-D subband coding of video," *IEEE Trans. Image Processing*, vol. 8, pp. 155–167, Feb. 1999.
[21] A. Said and W. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 243–250, June 1996.
[22] D. Marpe and H. L. Cycon, "Efficient pre-coding techniques for wavelet-based image compression," in *Proc. Picture Coding Symp.*, 1997, pp. 45–50.
[23] J. Rissanen and G. G. Langdon Jr, "Universal modeling and coding," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 12–23, Jan. 1981.
[24] J. Rissanen, "Universal coding, information, prediction, and estimation," *IEEE Trans. Inform. Theory*, vol. 30, pp. 629–636, July 1984.
[25] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression Standard*. New York: Van Nostrand Reinhold, 1993.
[26] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*. New York: McGraw-Hill, 1984, pp. 37–38.
[27] M. J. Weinberger, J. Rissanen, and R. B. Arps, "Application of universal context modeling to lossless compression of gray-scale images," *IEEE Trans. Image Processing*, vol. 5, pp. 575–586, Apr. 1996.
[28] A. N. Netravali and B. G. Haskell, *Digital Pictures, Representation and Compression*. New York: Plenum, 1988.
[29] J. Teuhola, "A compression method for clustered bit-vectors," *Inform. Processing Lett.*, vol. 7, pp. 308–311, Oct. 1978.
[30] R. Gallager and D. Van Voorhis, "Optimal source codes for geometrically distributed integer alphabets," *IEEE Trans. Inform. Theory*, vol. 21, pp. 228–230, Mar. IT-1975.
[31] M. Mrak, D. Marpe, and T. Wiegand, "A context modeling algorithm and its application in video compression," presented at the IEEE Int. Conf. Image Proc. (ICIP), Barcelona, Spain, Sept. 2003.
[32] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, and R. B. Arps, "An overview of the basic principles of the Q-coder adaptive binary arithmetic coder," *IBM J. Res. Dev.*, vol. 32, pp. 717–726, 1988.
[33] J. Rissanen and K. M. Mohiuddin, "A multiplication-free multialphabet arithmetic code," *IEEE Trans. Commun.*, vol. 37, pp. 93–98, Feb. 1989.
[34] P. G. Howard and J. S. Vitter, "Practical implementations of arithmetic coding," in *Image and Text Compression*, J. A. Storer, Ed. Boston, MA: Kluwer, 1992, pp. 85–112.
[35] D. Taubman and M. W. Marcellin, *JPEG2000 Image Compression: Fundamentals, Standards and Practice*. Boston, MA: Kluwer, 2002.
[36] D. Marpe and T. Wiegand, "A highly efficient multiplication-free binary arithmetic coder and its application in video coding," presented at the IEEE Int. Conf. Image Proc. (ICIP), Barcelona, Spain, Sept. 2003.
[37] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," in *Proc. IEEE Data Compression Conf.*, Snowbird, UT, 1996, pp. 202–211.
[38] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 688–703, July 2003.

Detlev Marpe (M'00) received the Dipl.-Math. degree (with highest honors) from the Technical University Berlin (TUB), Berlin, Germany, in 1990.

From 1991 to 1993, he was a Research and Teaching Assistant in the Department of Mathematics, TUB. Since 1994, he has been involved in several industrial and research projects in the area of still-image coding, image processing, video coding, and video streaming. In 1999, he joined the Fraunhofer–Institute for Telecommunications – Heinrich Hertz Institute (HHI), Berlin, Germany, where as a Project Leader in the Image Processing Department, he is currently responsible for projects focused on the development of advanced video coding and video transmission technologies. He has published more than 30 journal and conference articles in the area of image and video processing, and he holds several international patents. He has been involved in the ITU-T and ISO/IEC standardization activities for still image and video coding, to which he contributed more than 40 input documents. From 2001 to 2003, as an Ad-hoc Group Chairman in the Joint Video Team of ITU-T VCEG and ISO/IEC MPEG, he was responsible for the development of the CABAC entropy coding scheme within the H.264/AVC standardization project. His research interests include still image and video coding, image and video communication, computer vision, and information theory.

Mr. Marpe received the Prime Prize of the 2001 Multimedia Start-up Competition from the German Federal Ministry of Economics and Technology for his role as Cofounder of daViKo GmbH, a Berlin-based start-up company focused on developing server-less multipoint videoconferencing products for Intranet or Internet collaboration.

Heiko Schwarz received the Dipl.-Ing. degree in electrical engineering in 1996 and the Dr.-Ing. degree from the University of Rostock, Germany, in 2000.

In 1999, he joined the Fraunhofer–Institute for Telecommunications – Heinrich Hertz Institute (HHI), Berlin, Germany. Since then, he has contributed successfully to the ITU-T Video Coding Experts Group (VCEG) and the Joint Video Team (JVT) standardization efforts. His research interests include image and video compression, video communication, and signal processing.

Thomas Wiegand received the Dr.-Ing. degree from the University of Erlangen-Nuremberg, Germany, in 2000 and the Dipl.-Ing. degree in electrical engineering from the Technical University of Hamburg-Harburg, Germany, in 1995.

He is the Head of the Image Communication Group in the Image Processing Department, Fraunhofer–Institute for Telecommunications – Heinrich Hertz Institute (HHI), Berlin, Germany. During 1997 to 1998, he was a Visiting Researcher at Stanford University, Stanford, CA, and served as a Consultant to 8x8, Inc., Santa Clara, CA. From 1993 to 1994, he was a Visiting Researcher at Kobe University, Kobe, Japan. In 1995, he was a Visiting Scholar at the University of California at Santa Barbara, where he began his research on video compression and transmission. Since then, he has published several conference and journal papers on the subject and has contributed successfully to the ITU-T Video Coding Experts Group (ITU-T SG16 Q.6—VCEG)/ISO/IEC Moving Pictures Experts Group (ISO/IEC JTC1/SC29/WG11—MPEG)/Joint Video Team (JVT) standardization efforts and holds various international patents in this field. He has been appointed as the Associated Rapporteur of the ITU-T VCEG (October 2000), the Associated Rapporteur/Co-Chair of the JVT that has been created by ITU-T VCEG and ISO/IEC MPEG for finalization of the H.264/AVC video coding standard (December 2001), and the Editor of the H.264/AVC video coding standard (February 2002).