Entropy Coding in HEVC

Entropy Coding in HEVC

Vivienne Sze and Detlev Marpe

Abstract Context-Based Adaptive Binary Arithmetic Coding (CABAC) is a method of entropy coding first introduced in H.264/AVC and now used in the latest High Efficiency Video Coding (HEVC) standard. While it provides high coding efficiency, the data dependencies in H.264/AVC CABAC make it challenging to parallelize and thus limit its throughput. Accordingly, during the standardization of entropy coding for HEVC, both aspects of coding efficiency and throughput were considered. This chapter describes the functionality and design methodology behind CABAC entropy coding in HEVC.

1 Introduction

Context-Based Adaptive Binary Arithmetic Coding (CABAC) [51] is a form of entropy coding used in H.264/AVC [3] and also in HEVC [5]. Entropy coding is a lossless compression scheme that uses the statistical properties to compress data such that the number of bits used to represent the data is logarithmically proportional to the probability of the data. For instance, when compressing a string of characters, frequently used characters are each represented by a few bits, while infrequently used characters are each represented by many bits. From Shannon's information theory [72], when the compressed data is represented in bits $\{0,1\}$, the optimal average code length for a character with probability p is $-\log_2 p$.

Entropy coding is performed at the last stage of video encoding (and first stage of video decoding), after the video signal has been reduced to a series of syntax elements. Syntax elements describe how the video signal can be reconstructed at

Vivienne Sze
Massachusetts Institute of Technology (MIT) e-mail: sze@mit.edu

Detlev Marpe
Fraunhofer Institute for Telecommunications Heinrich Hertz Institute (HHI) e-mail: Detlev.Marpe@hhi.fraunhofer.de

the decoder. This includes the method of prediction (e.g. spatial or temporal prediction) along with its associated prediction parameters as well as the prediction error signal, also referred to as the residual signal. Note that in HEVC only the syntax elements belonging to the slice segment data are CABAC encoded. All other high level syntax elements are coded either with zero-order Exp-Golomb codes or fixed-pattern bit strings. Tab. 1 shows the syntax elements that are encoded with CABAC in HEVC and H.264/AVC. For HEVC, these syntax elements describe properties of the coding tree unit (CTU), prediction unit (PU), and transform unit (TU), while for H.264/AVC, the equivalent syntax elements have been grouped together along the same categories in Tab. 1. For a CTU, the related syntax elements describe the block partitioning of the CTU into coding units (CU), whether the CU is intra-picture (i.e. spatially) predicted or inter-picture (i.e., temporally) predicted, the quantization parameters of the CU, and the type (edge or band) and offsets for sample adaptive offset (SAO) in-loop filtering performed on the CTU. For a PU, the syntax elements describe the intra prediction mode or the motion data. For a TU, the syntax elements describe the residual signal in terms of frequency position, sign and magnitude of the quantized transform coefficients.

This chapter describes how CABAC entropy coding has evolved from H.264/AVC to HEVC. While high coding efficiency is important for reducing the transmission and storage cost of video, processing speed and area cost also need to be considered in the development of HEVC in order to handle the demand for higher resolutions and frame rates in future video coding systems. Accordingly, both coding efficiency and throughput improvement tools are discussed. Sect. 2 provides an overview of CABAC entropy coding. Sect. 3 explains the design considerations and techniques used to address both coding efficiency and throughput requirements. Sect. 4, Sect. 5, Sect. 6 and Sect. 7 describe how these techniques were applied to coding tree unit coding, prediction unit coding, transform unit coding and context initialization, respectively. Sect. 8 compares the coding efficiency, throughput and memory requirements of HEVC and H.264/AVC for both common conditions and worst case conditions.

2 CABAC Overview

The CABAC algorithm was originally developed within the joint H.264/AVC standardization process of ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Moving Picture Experts Group (MPEG). In a first preliminary version, the new entropy-coding method of CABAC was introduced as a standard contribution [45] to the ITU-T VCEG meeting in January 2001. CABAC was adopted as one of two alternative methods of entropy coding within the H.264/AVC standard. The other method specified in H.264/AVC was a low-complexity entropy-coding technique based on the usage of context-adaptively switched sets of variable-length codes, so-called Context-Adaptive Variable-Length Coding (CAVLC). Compared to CABAC, CAVLC offers reduced implementation cost at the price of lower compression ef-

Table 1: CABAC coded syntax elements in HEVC and H.264/AVC.

		HEVC	**H.264/AVC**
Coding Tree Unit (CTU) and Coding Unit (CU) **[Sect. 4]**	Coding Block and Prediction Block Structure, and Quantization Parameters **[Sect. 4.1–4.4]**	split_cu_flag, pred_mode_flag, part_mode, pcm_flag, cu_transquant_bypass_flag, cu_skip_flag, cu_qp_delta_abs, cu_qp_delta_sign_flag, end_of_slice_segment_flag, end_of_sub_stream_one_bit	mb_type, sub_mb_type, mb_skip_flag, mb_qp_delta, end_of_slice_flag, mb_field_decoding_flag
	Sample Adaptive Offset (SAO) Parameters **[Sect. 4.5]**	sao_merge_left_flag, sao_merge_up_flag, sao_type_idx_luma, sao_type_idx_chroma, sao_offset_abs, sao_offset_sign, sao_band_position, sao_eo_class_luma, sao_eo_class_chroma	n/a
Prediction Unit (PU) **[Sect. 5]**	Intra Prediction Mode Data **[Sect. 5.2]**	prev_intra_luma_pred_flag, mpm_idx, rem_intra_luma_pred_mode, intra_chroma_pred_mode	prev_intra4x4_pred_mode_flag, prev_intra8x8_pred_mode_flag, rem_intra4x4_pred_mode, rem_intra8x8_pred_mode, intra_chroma_pred_mode
	Motion Data **[Sect. 5.1]**	merge_flag, merge_idx, inter_pred_idc, ref_idx_l0, ref_idx_l1, abs_mvd_greater0_flag, abs_mvd_greater1_flag, abs_mvd_minus2, mvd_sign_flag, mvp_l0_flag, mvp_l1_flag	ref_idx_l0, ref_idx_l1, mvd_l0, mvd_l1
Transform Unit (TU) **[Sect. 6]**	Residual Data **[Sect. 6.1, 6.2, 6.3, 6.4, 6.5]**	rqt_root_cbf, split_transform_flag, cbf_luma, cbf_cb, cbf_cr, transform_skip_flag, last_sig_coeff_x_prefix, last_sig_coeff_y_prefix, last_sig_coeff_x_suffix, last_sig_coeff_y_suffix, coded_sub_block_flag, sig_coeff_flag, coeff_abs_level_greater1_flag, coeff_abs_level_greater2_flag, coeff_abs_level_remaining, coeff_sign_flag	coded_block_flag, coded_block_pattern, transform_size_8x8_flag, significant_coeff_flag, last_significant_coeff_flag, coeff_abs_level_minus1, coeff_sign_flag

ficiency. Typically, the bit-rate overhead for CAVLC relative to CABAC is in the range of 10–16% for standard definition (SD) interlaced material, encoded at Main Profile, and 15–22% for high definition (HD) 1080p material, encoded at High Profile, both measured at the same objective video quality and for the case that all other used coding tools within the corresponding H.264/AVC Profile remain the same [51, 50].

CABAC became also part of the first HEVC test model HM1.0 [55] together with the so-called low-complexity entropy coding (LCEC) as a follow-up of CAVLC. Later, during the HEVC standardization process, it turned out that the discrimination between high efficiency and low complexity was no longer necessary. Thus, CABAC in its improved form, both with respect to throughput speed and compression efficiency, became the single entropy coding method of the HEVC standard.

The basic design of CABAC involves the key elements of binarization, context modeling, and binary arithmetic coding. These elements are illustrated as the main algorithmic building blocks of the CABAC encoding block diagram in Fig. 1. Binarization maps the syntax elements to binary symbols (bins). Context modeling estimates the probability of each non-bypassed (i.e., regular coded) bin based on some specific context. Finally, binary arithmetic coding compresses the bins to bits according to the estimated probability.

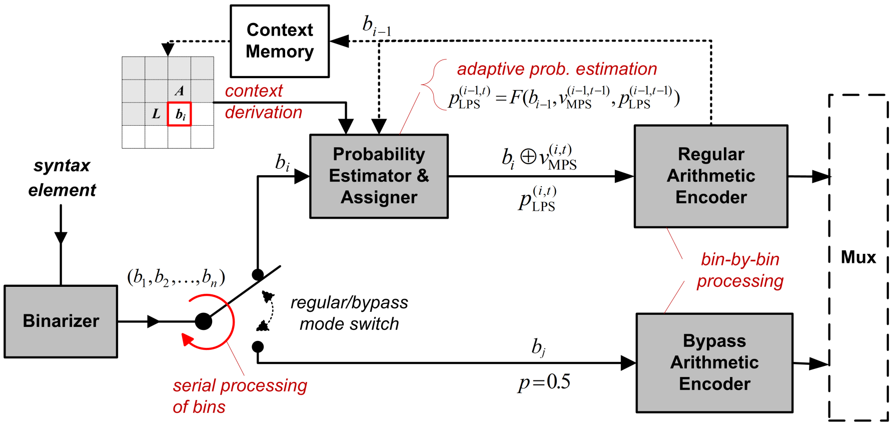

Context Memory
b_{i-1}
A
L
b_i
context derivation
adaptive prob. estimation
$p_{\text{LPS}}^{(i-1,t)} = F(b_{i-1}, v_{\text{MPS}}^{(i-1,t-1)}, p_{\text{LPS}}^{(i-1,t-1)})$
syntax element
b_i
Probability Estimator & Assigner
$b_i \oplus v_{\text{MPS}}^{(i,t)}$
$p_{\text{LPS}}^{(i,t)}$
Regular Arithmetic Encoder
Mux
$(b_1, b_2, \ldots, b_n)$
Binarizer
regular/bypass mode switch
bin-by-bin processing
b_j
$p=0.5$
Bypass Arithmetic Encoder
serial processing of bins

Fig. 1: CABAC block diagram (from the encoder perspective): Binarization, context modeling (including probability estimation and assignment), and binary arithmetic coding. In red: Potential throughput bottlenecks, as further discussed from the decoder perspective in Sect. 3.2.

2.1 Binarization

The coding strategy of CABAC is based on the finding that a very efficient coding of non-binary syntax-element values in a hybrid block-based video coder, like components of motion vector differences or transform-coefficient level values, can be achieved by employing a binarization scheme as a kind of preprocessing unit for the subsequent stages of context modeling and arithmetic coding. In general, a binarization scheme defines a unique mapping of syntax element values to sequences of binary symbols, so-called bins, which can also be interpreted in terms of a binary code tree. The design of binarization schemes in CABAC both for H.264/AVC and HEVC is based on a few elementary prototypes whose structure enables fast implementations and which are representatives of some suitable model-probability distributions.

Several different binarization processes are used in HEVC including *k*-th order truncated Rice (TRk), *k*-th order Exp-Golomb (EGk), and fixed-length (FL) binarization. Parts of these forms of binarization, including the truncated unary (TrU) scheme as the zero-order TRk binarization, were also used in H.264/AVC. These various methods of binarization can be explained in terms of how they would signal an unsigned value N. Examples are also provided in Tab. 2.

- Unary coding involves signaling a bin string of length $N+1$, where the first N bins are 1 and the last bin is 0. The decoder searches for a 0 to determine when the syntax element is complete. For the TrU scheme, truncation is invoked for the largest possible value cMax[1] of the syntax element being decoded.
- *k*-th order truncated Rice is a parameterized Rice code that is composed of a prefix and a suffix. The prefix is a truncated unary string of value $N >> k$, where the largest possible value is cMax. The suffix is a fixed length binary representation of the least significant bins of N; k indicates the number of least significant bins. Note that for k= 0, the truncated Rice is equal to the truncated unary binarization.
- *k*-th order Exp-Golomb code is proved to be a robust, near-optimal prefix-free code for geometrically distributed sources with unknown or varying distribution parameter. Each codeword consists of a unary prefix of length $l_N + 1$ and a suffix of length $l_N + k$, where $l_N = \lfloor \log_2((N >> k)+1) \rfloor$ [51].
- Fixed-length code uses a fixed-length bin string with length $\lceil \log_2(\mathrm{cMax}+1) \rceil$ and with most significant bins signaled before least significant bins.

The binarization process is selected based on the type of syntax element. In some cases, binarization also depends on the value of a previously processed syntax element (e.g. binarization of `coeff_abs_level_remaining` depends on the previously decoded coefficient levels) or slice parameters that indicate if certain modes are enabled (e.g. binarization of partition mode, so-called `part_mode`, depends on whether asymmetric motion partition is enabled). The majority of the syntax elements use the binarization processes as listed above, or some combination of them (e.g. `cu_qp_delta_abs` uses TrU (prefix) + EG0 (suffix) [94]). However, certain

[1] cMax is defined by the standard for each relevant type of syntax element.

Table 2: Examples of different binarizations.

N	**Unary (U)**	**Truncated Unary (TrU)**	**Truncated Rice (TRk)**	**Exp-Golomb (EGk)**	**Fixed-Length (FL)**
		cMax=7	$k = 1$; cMax=7	$k = 0$	cMax=7
0	0	0	00	1	000
1	10	10	01	010	001
2	110	110	100	011	010
3	1110	1110	101	00100	011
4	11110	11110	1100	00101	100
5	111110	111110	1101	00110	101
6	1111110	1111110	1110	00111	110
7	11111110	1111111	1111	0001000	111

syntax elements (e.g. `part_mode` and `intra_chroma_pred_mode`) use custom binarization processes.

During the HEVC standardization process, special attention has been put on the development of an adequately designed binarization scheme for absolute values of transform coefficient levels. In order to guarantee a sufficiently high throughput, the goal here was the maximization of bypass-coded bins under the constraint of not sacrificing coding efficiency too much. This was accomplished by making the binarization scheme adaptive based on previously coded transform coefficient levels. More details on that are given in Sect. 6.5.

2.2 Context Modeling, Probability Estimation and Assignment

By decomposing each non-binary syntax element value into a sequence of bins, further processing of each bin value in CABAC depends on the associated coding-mode decision, which can be either chosen as the regular or the bypass mode (as described in Sect. 2.3). The latter is chosen for bins, which are assumed to be uniformly distributed and for which, consequently, the whole regular binary arithmetic encoding (and decoding) process is simply bypassed. In the regular coding mode, each bin value is encoded by using the regular binary arithmetic coding engine, where the associated probability model is either determined by a fixed choice, based on the type of syntax element and the bin position or bin index (binIdx) in the binarized representation of the syntax element, or adaptively chosen from two or more probability models depending on the related side information (e.g. spatial neighbors as illustrated in Fig. 1, component, depth or size of CU/PU/TU, or position within TU). Selection of the probability model is referred to as context modeling. As an important design decision, the latter case is generally applied to the most frequently observed bins only, whereas the other, usually less frequently observed bins, will be treated using a joint, typically zero-order probability model. In this way, CABAC enables selective adaptive probability modeling on a sub-symbol level, and hence, provides an efficient instrument for exploiting inter-symbol redundancies at significantly reduced overall modeling or learning costs. Note that for both the fixed and the adaptive case, in principle, a switch from one probability model to another can occur between any two consecutive regular coded bins. In general, the design of context models in CABAC reflects the aim to find a good compromise between the conflicting objectives of avoiding unnecessary modeling-cost overhead and exploiting the statistical dependencies to a large extent.

The parameters of probability models in CABAC are adaptive, which means that an adaptation of the model probabilities to the statistical variations of the source of bins is performed on a bin-by-bin basis in a backward-adaptive and synchronized fashion both in the encoder and decoder; this process is called *probability estimation*. For that purpose, each probability model in CABAC can take one out of 126 different states with associated model probability values p ranging in the interval $[0.01875, 0.98125]$. The two parameters of each probability model are stored as 7-bit entries in a context memory: 6 bits for each of the 63 probability states representing the model probability p_{LPS} of the least probable symbol (LPS) and 1 bit for ν_{MPS}, the value of the most probable symbol (MPS). The probability estimator in CABAC is based on a model of "exponential aging" with the following recursive probability update after coding a bin b at time instance t:

$$p_{\text{LPS}}^{(t+1)} = \begin{cases} \alpha \cdot p_{\text{LPS}}^{(t)}, & \text{if } b = \nu_{\text{MPS}}, \text{ i.e., an MPS occurs} \\ 1 - \alpha \cdot (1 - p_{\text{LPS}}^{(t)}), & \text{otherwise.} \end{cases} \tag{1}$$

Here, the choice of the scaling factor α determines the speed of adaptation: A value of α close to 1 results in a slow adaptation ("steady-state behavior"), while faster adaptation can be achieved for the non-stationary case with decreasing α. Note that this estimation is equivalent to using a sliding window technique [65, 9] with window size $W_\alpha = (1-\alpha)^{-1}$. In the design of CABAC, Eq. (1) has been used together with the choice of

$$\alpha = \left(\frac{0.01875}{0.5}\right)^{\frac{1}{63}} \text{ with } \min_t p_{\text{LPS}}^{(t)} = 0.01875, \tag{2}$$

and a suitable quantization of the underlying LPS-related model probabilities into 63 different states, to derive a finite-state machine (FSM) with tabulated transition rules [51]. This table-based probability estimation method was unchanged in HEVC, although some proposals for alternative probability estimators [6, 78] have shown average bitrate savings of 0.8–0.9%, albeit at higher computational costs.

Each probability model in CABAC is addressed using a unique context index (ctxIdx), either determined by a fixed assignment or computed by the context derivation logic by which, in turn, the given context model is specified. A lot of effort has been spent during the HEVC standardization process to improve the model assignment and context derivation logic both in terms of throughput and coding efficiency. More details on the specific choice of context models for selected syntax elements in HEVC are given in Sect. 4–6.

2.3 *Multiplication-Free Binary Arithmetic Coding: The M Coder*

Binary arithmetic coding, or arithmetic coding in general, is based on the principle of recursive interval subdivision. An initially given interval represented by its lower bound (base) L and its width (range) R is subdivided into two disjoint subintervals: one interval of width

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

which is associated with the LPS, and the dual interval of width $R_{\text{MPS}} = R - R_{\text{LPS}}$, which is assigned to the MPS. Depending on the binary value to encode, either identified as LPS or MPS, the corresponding subinterval is then chosen as the new coding interval. By recursively applying this interval-subdivision scheme to each bin b_j of a given sequence $\mathbf{b} = (b_1, b_2, \ldots, b_N)$ of bins, the encoder finally determines a value $c_{\mathbf{b}}$ in the subinterval $[L^{(N)}, L^{(N)} + R^{(N)})$ that results after the N^{th} interval subdivision process. The (minimal) binary representation of $c_{\mathbf{b}}$ is the arithmetic code of the input bin sequence **b**. To ensure that finite-precision registers are sufficient to represent $R^{(j)}$ and $L^{(j)}$ for all $j \in \{1, 2, \ldots, N\}$, a renormalization operation is required, whenever $R^{(j)}$ falls below a certain limit after one or more interval subdivision process(es). By renormalizing $R^{(j)}$, and accordingly $L^{(j)}$, the leading bits of the arithmetic code can be output as soon as they are unambiguously identified.

On the decoder side, the sequence of encoded binary values can be easily recovered by tracking the interval subdivision, including renormalization, according to Eq. (3) step-by-step and by comparing the bounds of both subintervals to the transmitted value representing the final subinterval. Note that the width $R^{(N)}$ of the final subinterval is proportional to the product $\prod_{j=1}^{N} p(b_j)$ of the individual model probability $p(b_j)$ assigned to the bins b_j of the bin sequence, such that for signaling the final subinterval, the lower bound of the empirical entropy of the bin sequence given by $-\log_2 \prod_{j=1}^{N} p(b_j) = -\sum_{j=1}^{N} \log_2 p(b_j)$ is approximately achieved.

From a practical implementation point of view, the most costly operation involved in binary arithmetic coding is given by the multiplication in Eq. (3). Even worse, if probability estimation is based on a simple scaled-count estimator using scaled cumulative frequency counts of bins, this operation may even involve an integer division. A solution to this problem was already proposed during the H.264/AVC standardization process by using a design of a family of multiplication-free binary arithmetic coders, which later became known as the *modulo coder* (M coder) [47, 54]. The main innovative features of this design are given by a table-based interval subdivision coupled with the above-mentioned FSM-based probability estimation as well as a fast bypass coding mode. The former, which is also the basis of what is called the *regular coding mode* of the M coder, will be briefly reviewed next, followed by a short discussion of the latter aspect.

2.3.1 Regular Coding Mode

The basic idea of the M-coder approach of interval subdivision is to quantize the range of possible interval widths induced by renormalization into a small number of K cells. To further simplify matters, a uniform quantization with $K = 2^{\kappa}$ is assumed to be performed, resulting in a set $\mathbf{Q} = \{Q_0, Q_1, \cdots, Q_{K-1}\}$ of representative interval widths. Together with the representative set of LPS-related probability values of the FSM given by $\mathbf{P} = \{p_0, p_1, \cdots, p_{N-1}\}$, this quantization enables the approximation of the multiplication on the right-hand side of Eq. (3) by means of a table of $K \times N$ pre-calculated product values $\{Q_k \cdot p_n \,|\, 0 \leq k < K; 0 \leq n < N\}$ in a suitable chosen integer precision. The entries of the corresponding 2-D lookup table *TabRangeLPS* are addressed by the (probability) state index n and the quantization cell index $k(R)$ related to the given value of the interval range R. Computation of $k(R)$ is easily carried out by a concatenation of a bit shift and a bit-masking operation, where the latter can be interpreted as a *modulo operation* using the operand $K = 2^{\kappa}$, hence the naming of the family of coders.

In the context of H.264/AVC, the optimal empirical choice of the free parameters $\kappa = 2$ and $N = 64$ was determined under the constraint of a maximum table size of $2^{\kappa} \cdot N \leq 256$ bytes for the lookup table *TabRangeLPS* with each of its entries being represented with 8 bits. This specific M-coder design of using a lookup table *TabRangeLPS* with 4×64 entries was also adopted for HEVC. Please note that by choosing a value of $\kappa = 0$, the 2-D table *TabRangeLPS* degenerates to a 1-D table, where for all possible values of R only one single representative value is used for the approximation of $p_n \cdot R$. This choice is equivalent to the subinterval division operation performed in the Q coder and its derivatives of QM and MQ coder, as being standardized in JBIG, JPEG, and JPEG2000. Thus, the M-coder design can be interpreted as a generalization of the Q-coder family[2]. Compared to the QM/MQ coder, the M coder, being configured as in H.264/AVC and HEVC, achieves an increase in throughput of 18%, while at the same time it provides bit-rate savings of 2–4%, when evaluated in the CABAC environment of H.264/AVC [54]. Interestingly, the throughput improvements of the M coder can be largely attributed to its unique bypass functionality, as being reviewed in the next subsection, while its use of a larger lookup table for interval subdivision generates the main effects in coding-efficiency gain; however, this increased table size can also adversely affect the overall throughput gain of the M coder.

2.3.2 Bypass Coding Mode

As already mentioned, most of the throughput improvements of the M coder relative to the Q-coder technology can be attributed to its second innovative feature, which is given by a bypass of the probability estimation for approximately uniform

[2] Please note that apart from the interval subdivision aspect there are some subtle technical differences between (and also within) the coder families, such as concerning, e.g., probability estimation, conditional exchange, carry-over handling, and termination.

distributed bins. In addition, the interval subdivision is substituted by a hard-wired equipartition in this so-called bypass coding mode. In this way, the whole encoding/decoding process (including renormalization) can be realized by nothing more than a bit shift, a comparison, and for half of the symbols an additional subtraction.

Bypass coding has become an even more important feature during the HEVC standardization process. While in H.264/AVC bypass coding was mainly used for signs and least significant bins of absolute values of quantized transform coefficients, in HEVC the majority of possible bin values is handled through the bypass coding mode. As noted above, this is also a consequence of carefully designed binarization schemes, which already serve as a kind of near-optimal prefix-free codes of the corresponding syntax elements.

2.3.3 Fast Renormalization

One of the major throughput bottlenecks in any arithmetic encoding and decoding process is given by the renormalization procedure. Renormalization in the M coder is required whenever the new interval range R after interval subdivision no longer stays within its admissible domain. Each time a renormalization operation must be carried out, one or more bits can be outputted at the encoder or, equivalently, have to be read by the decoder. This process, as it is specified in H.264/AVC and HEVC, is performed bit-by-bit and is controlled by some conditional branches to check each time if further renormalization loops are required. Both conditional branching and bitwise processing, however, constitute considerable obstacles to a sufficiently high throughput.

As a mitigation of this problem, a fast renormalization policy for the M coder was proposed in [50]. By replacing the conventionally bitwise performed operations in the regular coding mode with byte-wise or word-wise processing, a considerably increased decoder throughput of around 25% can be achieved. The corresponding non-normative, fully standard-compliant changes were integrated into the reference software implementations of both H.264/AVC and HEVC. For more details, please refer to [50, 49].

2.3.4 Termination

For termination of the arithmetic codeword in the M coder a special, non-adapting probability state is reserved. The corresponding probability state index is given by $n = 63$ and the corresponding entries of *TabRangeLPS* deliver a constant value of $R_{\text{LPS}} = 2$. As a consequence, for each terminating syntax element, such as `end_of_slice_segment_flag`, `end_of_sub_stream_one_bit`, or `pcm_flag`, 7 bits of output are generated in the renormalization process. Two more bits are needed to be flushed in order to properly terminate the arithmetic codeword. Note that the least significant bit in this flushing procedure, i.e., the last written bit at the encoder is always equal to 1 and thus, represents the so-called `rbsp_stop_one_bit`. Before packaging of the bitstream, the arithmetic codeword is filled up for byte alignment with zero-valued alignment bits.

3 Design Considerations

Most of the proposals submitted to the joint Call for Proposals on HEVC in April 2010 already included some form of advanced entropy coding. Some of those techniques were based on improved versions of CAVLC or CABAC, others were using alternative methods of statistical coding, such as V2V (variable-to-variable) codes [34] or PIPE (probability interval partitioning entropy) codes [53, 52, 102], and a third category introduced increased capabilities for parallel processing on a bin level [85], syntax element level [91, 92], or slice level [105, 33, 29]. In addition, improved techniques for coding of transform coefficients, such as zero-tree representations [7], alternate scanning schemes [41], or template-based context models [61, 102], were proposed.

After an initial testing phase of video coding technology from the best performing HEVC proposals, it was decided to start the first HEVC test model (HM1.0) [55] with two alternate configurations similar to what was given for entropy coding in H.264/AVC: a high efficiency configuration based on CABAC and a low-complexity configuration based on LCEC as a CAVLC surrogate. Interestingly enough, the CABAC-based entropy coding of HM1.0 already included techniques for improving both coding efficiency and throughput relative to its H.264/AVC-related predecessor. To be more specific, a template-based context modeling scheme for larger transform block sizes [61, 46] and a parallel context processing technique for selected syntax elements of transform coefficient coding [15] became already part of HM1.0. During the subsequent collaborative HEVC standardization phase, more techniques covering both aspects of coding efficiency and throughput were integrated, as will be discussed in more details in the following.

While CABAC inherently is targeting at high coding efficiency, its data dependencies can cause it to be a throughput bottleneck, especially at high bit rates as was already analyzed in the context of H.264/AVC [93]. This means that, without any further provision, it might have been difficult to support the growing throughput requirements for future video codecs. Furthermore, since high throughput can be traded-off for power savings using voltage scaling [19], the serial nature of CABAC may limit the battery life for video codecs that reside on mobile devices. This limitation is a critical concern, as a significant portion of video codecs today are running on battery-operated devices. Accordingly, both coding efficiency and throughput improvement tools as well as the trade-off between these two requirements were investigated in the standardization of entropy coding for HEVC. The trade-off between coding efficiency and throughput comes from the fact that, in general, dependencies are a result of removing redundancy which, in turn, improves coding efficiency; however, increasing dependencies usually makes parallel processing more difficult which, as a consequence, may degrade throughput. This section describes the various techniques used to improve both coding efficiency and throughput of CABAC entropy coding for HEVC.

3.1 Brief Summary of HEVC Block Structures and CABAC Coding Efficiency Improvements

In the evolutionary process from H.264/AVC to HEVC, improved coding efficiency for CABAC entropy coding was addressed in a number of proposals, such as [102, 106, 28]. The majority of coding-efficiency related CABAC proposals in the HEVC standardization process was oriented towards transform coefficient coding, since at medium to high bit rates the dominant part of bits is consumed by syntax elements related to residual coding. As a consequence, this subsection will focus on considerations that were made with regards to the specific CABAC design for those syntax elements. Note, however, that due to the more consistent design of HEVC in terms of tree structures for both partitioning of prediction blocks and transform blocks, special care has also been taken to ensure an efficient modeling and coding of the corresponding tree structuring elements. In addition, for new coding tools in HEVC, such as block merging and sample-adaptive offset (SAO) in-loop filtering, additional assignments of binarization and context modeling schemes were needed.

Transform coding in HEVC is based on a tree-structured variable block-size approach with the corresponding quadtree structure referred to as *residual quadtree* (RQT) [102, 46]. RQTs are nested into the leaves of another quadtree, the so-called *coding quadtree* (CQT), which determines the subdivision of each block of $2^N \times 2^N$ luma samples, referred to as a coding tree block (CTB) [102, 46]. The block partitioning for both prediction and transform coding is the same for luma and chroma picture component samples[3], and hence, a common coding and residual quadtree syntax is used to signal the partitioning. As a result, the blocks of luma and chroma samples and associated syntax elements are grouped together in a so-called *unit*.

A transform unit (TU) aggregates the transform blocks (TBs) of luma and chroma samples as well as the syntax elements used to represent the associated transform coefficient levels. Each TU and the related luma and two chroma TBs are determined as a leaf of the corresponding RQT. Supported TB sizes for both luma and chroma are in the range from 4×4 to 32×32 samples, where the corresponding core transforms are separable applications of a fixed-point approximation of the 1-D Discrete Cosine Transform (DCT) for dyadically increasing lengths from 4 to 32 points [30]. An exception is given for 4×4 luma TBs of residual signals resulting from intra-picture predicted blocks, where instead of the DCT-like core transform a separable fixed-point approximation of the 1-D Discrete Sine Transform (DST) is used [101].

[3] There is one exception to this general rule in HEVC, which is discussed in more detail in Chap. “Block Structures and Parallelism Features in HEVC”.

Note that a prediction unit (PU) aggregates the prediction blocks (PBs) of luma and chroma samples and the associated syntax elements like motion data. A coding unit (CU) encapsulates the luma and chroma coding block (CB) samples and the so-called prediction mode, i.e., the decision whether the corresponding samples are coded using intra-picture or inter-picture prediction, as well as some additional syntax elements. On the top level of the hierarchy, a coding tree unit (CTU) comprises the CTBs of luma and chroma samples, the associated CQT syntax structure and all CUs at the CQT leaves.

3.1.1 Coefficient grouping into subblocks

Given the larger variety of TB sizes, one of the primary goals of CABAC entropy coding for transform coefficient data in HEVC was to achieve a design that uses for all block sizes as much of the same logic and the same procedures as possible. Although at first glance this objective seems to be somehow unrelated to coding efficiency, it turns out that at least one particular element leading to such a unified design is also crucial for achieving high coding efficiency. This coding element is given by the grouping of coefficients into so-called subblocks of size 4×4 for transform blocks with size greater than 4×4. Subblocks were first proposed in [102, 46, 61] and became part of HM1.0. In the subsequent HEVC development process, their use was iteratively refined and extended in a way as will be explained in more detail in Sect. 6.

3.1.2 Hierarchy of significance flags

Since for most common coding conditions, a large portion of transform coefficients is quantized to zero, or equivalently, the representation of the residual signal in the DCT-/DST-like basis functions is supposed to be sparse, a hierarchical structured set of four different significance flags[4] is introduced in HEVC to reduce the number of individual significance flags to be transmitted. This hierarchy of syntax elements also reflects the hierarchical processing of TBs within the RQT as well as the processing of subblocks within a given TB.

The use of so-called *coded block flags* (CBF), indicating the occurrence of significant, i.e., nonzero transform coefficients in a TB, was already part of H.264/AVC CABAC-based residual coding. In HEVC, this concept was extended to also cover the RQT root on the top level of the hierarchy as well as the subblock on a lower level of the hierarchy. Consequently, there are a `rqt_root_cbf`, at least for RQT roots in inter-predicted CUs, `cbf_luma`, `cbf_cb`, and `cbf_cr` for the visited TBs of the 3 color components, and a `coded_sub_block_flag` (CSBF) for each visited subblock in a TB. On the lowest level of the hierarchy, for each visited subblock

[4] Note that the term “significance flag” is interpreted here and in the following in a much broader sense than originally used in the context of H.264/AVC.

a so-called *significance map* indicates the location of nonzero coefficients for each scan position in a subblock.

This hierarchy of significance flags is complemented by the syntax elements indicating the last significant scan position in a TB, which somehow serve as an entry point into each significant TB and which is equivalent to signaling the insignificance of a partial area of a TB. The latter concept differs from H.264/AVC, where for each `significant_coeff_flag` (SIG) with a value of one, a `last_significant_coefficient_flag` (LAST) is signaled indicating if the current scan position is the last nonzero coefficient inside the TB. Note that this latter signaling scheme is equivalent to using a TrU binarization (with inverted bin values) for the number of nonzero coefficients in a TB, such that each bin of the resulting bin sequence is intertwined with the corresponding nonzero significance flag. This design aspect of mixing two flags on a bin level in H.264/AVC was later found to be critical in terms of throughput, as will be discussed in Sect. 6.

3.1.3 Context modeling for coding of significance flags

Particular care has been taken to properly specify the context models for coding of significance flags. For instance, modeling of the CBF is based on the RQT depth, while that for the CSBF is using neighboring CSBF information. For coding of the significance map, which typically consumes most of the bits in HEVC transform coding, additional dependencies between neighboring elements have been exploited, at least for TBs larger than 4×4. Initially, for that purpose a local template was proposed [102, 46, 61] and adopted for HM1.0. Although this design provides high coding efficiency, it introduces some critical data dependencies. As a solution to this problem, a combination of position-based information (as used in H.264/AVC) and template-based neighborhood information was finally adopted for context modeling of significance map entries [43, 77]. This particular example also illustrates how both aspects of coding efficiency and throughput were considered during the HEVC standardization process in a balanced way. More on the throughput aspects is given in the next subsection, while the details of context modeling for all syntax elements related to residual coding are provided in Sect. 6.

3.2 CABAC Throughput Bottlenecks

CABAC, as originally designed for H.264/AVC and also, as initially selected for the HEVC standardization starting point in HM1.0, has some serious throughput issues (particularly for decoder implementations at higher bit rates) [93, 80]. The throughput of CABAC is determined based on the number of binary symbols (bins) that it can process per second. The throughput can be improved by increasing the number of bins that can be processed in a cycle. However, the data dependencies in CABAC make processing multiple bins in parallel difficult and costly to achieve.

These dependencies result in feedback loops in the CABAC decoder as shown in Fig. 2, and can be described as follows:

1. The updated range is fed back for recursive interval subdivision.
2. The updated context is fed back for probability estimation.
3. The context modeler selects the probability model based on the type of syntax element and, as already noted above, for selected syntax elements, based on some derivation process that involves other previously decoded bin values or other relevant side information. At the decoder, for non-binary syntax elements, the decoded bin value is fed back to determine whether to continue processing the same syntax element or to switch to another syntax element. If a switch occurs, the value of the decoded bin may also be used to determine which syntax element to decode next.
4. The context modeler may also select the probability model based on the bin position in the syntax element (binIdx). At the decoder, the decoded bin value is fed back to determine whether to increment binIdx and continue to decode the current syntax element, or set binIdx equal to 0 and switch to another syntax element.

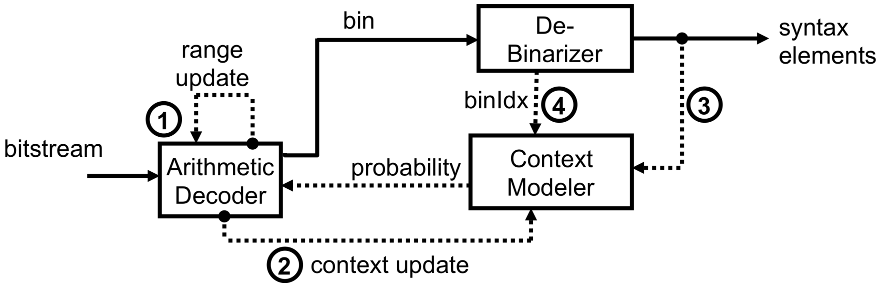


Fig. 2: Three key operations in CABAC (from a decoder perspective): Binarization, Context Modeling/Selection and (Binary) Arithmetic Coding. Feedback loops in the decoder are highlighted with dashed lines.

Note that the feedback loops have different degrees of impact on throughput. The range update (1) and context update (2) feedback loops are simpler than the context modeling loops (3, 4) and thus do not affect throughput as severely. If the context of a bin depends on the value of another bin being decoded in parallel, then speculative computations are required, which increases area cost and critical path delay [92]. The amount of speculation can grow exponentially with the number of parallel bins, which limits the throughput that can be achieved [80]. Fig. 3 shows an example of the speculation tree for significance map in H.264/AVC. Thus the throughput bottleneck is primarily due to the context modeling dependencies.

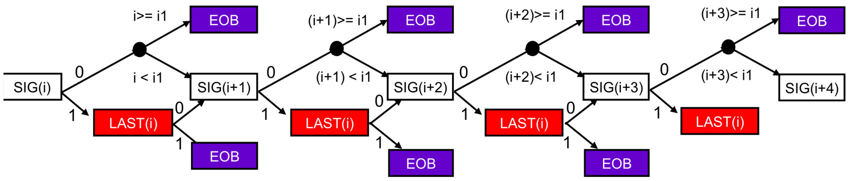


Fig. 3: Context speculation required to achieve 5× parallelism when processing the significance map in H.264/AVC. Notation: i = coefficient position; i1 = MaxNumCoeff(BlockType)−1; EOB = end of block; SIG = `significant_coeff_flag`; LAST = `last_significant_coeff_flag`.

3.3 Summary of Techniques for CABAC Throughput Improvements

Several techniques were used to improve the throughput of CABAC in HEVC [89]. There was a lot of effort spent in determining how to use these techniques with minimal coding loss. They were applied to various parts of entropy coding in HEVC and will be referred to throughout the rest of this chapter.

3.3.1 Reduce regular coded bins

The throughput is limited for regular coded bins due to the data dependencies described in Sect. 3.2. However, it is easier to process bypass coded bins in parallel since they do not have the data dependencies related to context modeling (i.e. feedback loops 2, 3 and 4 in Fig. 2). In addition, arithmetic coding for bypass bins is simpler as it only requires a right shift versus a table look up for regular coded bins. Thus, the throughput can be improved by reducing the number of regular coded bins and using bypass coded bins instead [62, 58, 60, 21].

3.3.2 Group bypass coded bins

Multiple bypass bins can be processed in the same cycle only if they occur *consecutively* within the bitstream. Thus, bins should be reordered such that bypass coded bins are grouped together in order to increase the likelihood that multiple bins are processed per cycle [88, 67, 23].

3.3.3 Group bins with same context

Processing multiple regular coded bins in the same cycle often requires speculative calculations for context modeling. The amount of speculative computations increases if bins using different contexts and context modeling logic are interleaved, since numerous combinations and permutations must be accounted for. Thus, to reduce speculative computations, bins should be reordered such that bins with the same contexts and context modeling logic are grouped together so that they are likely to be processed in the same cycle [15, 14, 73]. This also reduces context switching resulting in fewer memory accesses, which also increases throughput and reduces power consumption. This technique was first introduced in [15] and was referred to as parallel context processing (PCP) throughout the standardization process.

3.3.4 Reduce context modeling dependencies

Speculative computations are required for multiple bins per cycle decoding due to the dependencies in the context modeling. For instance, this is an issue when the context modeling for the next bin depends on the decoded value of the current bin. Reducing these dependencies simplifies the context modeling logic and reduces the amount of speculative calculations required to process multiple bins in parallel [80, 86, 22].

3.3.5 Reduce total number of bins

In addition to increasing the throughput, it is desirable to reduce the workload itself by reducing the total number of bins that need to be processed. This can be achieved by changing the binarization, inferring the value of some bins[5], and sending higher level flags to avoid signaling redundant bins [62, 18, 57].

3.3.6 Reduce parsing dependencies

As parsing with CABAC may constitute a throughput bottleneck, it is important to minimize any dependency on other video decoding processes, which could cause CABAC to stall or may even prevent a successful parsing process in case of picture loss due to transmission errors [108, 79, 12] (see Sect. 5.1.1). Ideally the parsing process should be decoupled from all other decoding processes, which actually is the case for CABAC in H.264/AVC. Decoupling parsing from the sample reconstruction process is also important when entropy decoupling is used, i.e., when a large frame level buffer is inserted between the entropy decoder and the rest of the decoder to absorb the variance in the bit-rate and pixel-rate workloads, respectively.

[5] The benefit of inferring bins must be traded-off with a potential increase in context selection complexity.

3.3.7 Reduce memory requirements

Memory accesses often contribute to the critical path delay. Thus, reducing memory storage requirements is desirable as fewer memory accesses increases throughput as well as reduces implementation cost and power consumption [81, 95].

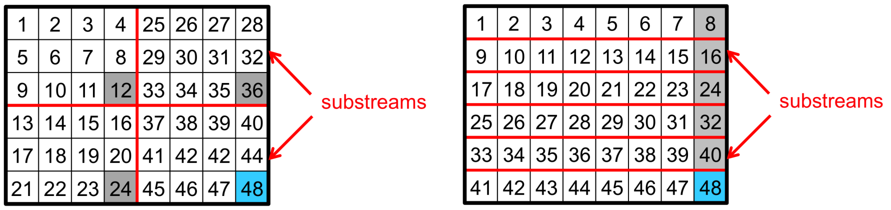
4 Coding Tree Unit and Coding Unit Syntax Elements

In HEVC, a picture is partitioned into a regular grid of disjoint square blocks of $2^N \times 2^N$ luma samples and, in case of 4:2:0 color sampling, corresponding square blocks of $2^{N-1} \times 2^{N-1}$ chroma samples. The parameter $N = 4, 5$, or 6 can be chosen by the encoder and transmitted in the sequence parameter set (SPS), such that the corresponding coding tree units represent luma CTBs of size 16×16, 32×32, or 64×64 samples, respectively. The CTU syntax elements describe how the corresponding CTBs can be further partitioned into smaller coding blocks by use of the coding quadtree and how the method of sample adaptive offset (SAO) in-loop filtering is performed on the reconstructed luma and chroma samples belonging to the CTU.

Within a picture, an integer number of CTUs can be grouped into a slice. Each slice itself consists of one (leading) independent slice segment and zero or more subsequently ordered dependent slice segments. A flag called `end_of_slice_segment_flag` is sent to indicate the last CTU in a slice segment. In addition, tiles and wavefront parallel processing, which are introduced in Chap. "Block Structures and Parallelism Features in HEVC", can be used to fragment the slice segment into multiple substreams[6], each being represented by its own CABAC codeword. Therefore, if `end_of_slice_segment_flag` indicates that it is not the last CTU in a slice segment, a flag called `end_of_sub_stream_one_bit` is used to indicate whether it is the last CTU of the corresponding substream.[7] An example of this is illustrated in Fig. 4. Both `end_of_slice_segment_flag` and `end_of_sub_stream_one_bit` are coded using the terminating mode of the arithmetic coding engine. This is required since at the end of a slice segment or a substream, the arithmetic coding engine must be flushed and the resulting CABAC codeword must be byte aligned before, at least in the former case, inserting the startcode for the next slice or entry point for the next slice segment. Fig. 5 shows an example of the locations of CABAC termination within a bitstream.

[6] Slice segments can also be used to fragment tiles and wavefronts into substreams.

[7] Note that the value of `end_of_sub_stream_one_bit` is always 1 and it is only sent for the last CTU of a substream.



1	2	3	4	25	26	27	28
5	6	7	8	29	30	31	32
9	10	11	12	33	34	35	36
13	14	15	16	37	38	39	40
17	18	19	20	41	42	42	44
21	22	23	24	45	46	47	48

(a) *Tiles*: CTU 12, 24, and 36 have (0, 1); CTU 48 (1, not signaled); and the rest of the CTUs have (0,0).

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48

(b) *Wavefront parallel processing*: CTU 8, 16, 24, 32 and 40 have (0, 1); CTU 48 (1, not signaled); and the rest of the CTUs have (0,0).

Fig. 4: These two examples illustrate which CTUs are terminated when slice segments are divided into substreams using tiles and wavefront parallel processing. Values of (end_of_slice_segment_flag, end_of_sub_stream_one_bit) are given for each configuration.

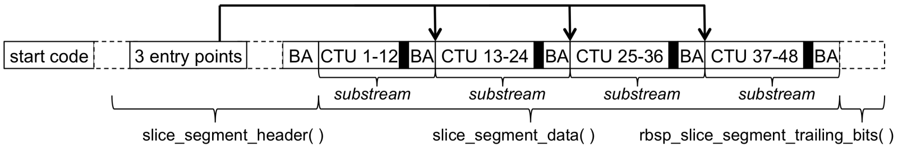


Fig. 5: Ordering of the bitstream for the tiles example in Fig. 4a. CABAC needs to be terminated before byte alignment (*BA*) as shown by the black boxes. Entry points for substreams are sent in *slice_segment_header()*.

4.1 Coding Block Structure

The coding block structure is determined by the coding quadtree which is signaled by a flag called split_cu_flag at each of its nodes to indicate whether a given coding block should be further subdivided into four smaller CBs. There is a strong spatial correlation between the chosen CQT depth of neighboring CBs, i.e., the block sizes of neighboring CBs, thus the context selection for split_cu_flag depends on the relative depth of the top and left neighboring CBs compared to that of the current CB. Note that in H.264/AVC the partitioning information is sent together with other data as aggregated syntax elements mb_type and sub_mb_type with different ranges of allowed values and hence different binarization schemes for different slices.[8] This kind of aggregating different information in one single syntax element is mostly due to historical reasons, reflecting the circumstances that earlier video coding standards (including H.264/AVC) were designed under the regime of VLC-based entropy coding, where alphabet extensions are used to circumvent

[8] In H.264/AVC, mb_type and sub_mb_type are used to represent the following equivalent information in HEVC: split_cu_flag, pred_mode_flag, part_mode, pcm_flag, inter_pred_idc, coded block pattern (cbf_luma, cbf_cr, cbf_cb) and intra prediction mode for 16×16 intra-coded PU.

the lower bound of 1 bit per symbol. Thus, by allowing the signaling of a coding quadtree structure with a one-bin syntax element, i.e., the `split_cu_flag` at each node, HEVC is much more flexible and allows many more coding and prediction block structures than H.264/AVC, even when choosing a CTB size of 16 × 16 luma samples and ignoring the fact that HEVC doesn't allow for inter-predicted 4 × 4 luma blocks, as discussed in Chap. "Block Structures and Parallelism Features in HEVC".

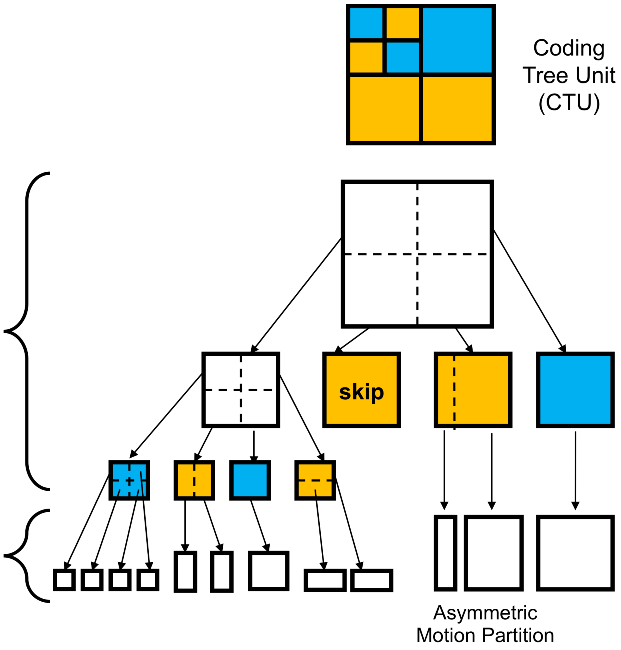


Fig. 6: A coding tree unit is subdivided into CUs along the associated coding quadtree. Each resulting CU may be further subdivided into PUs. The intra-coded CUs are in blue while inter-coded CUs are in orange. Note that the figure only shows the corresponding CTB, CBs, and PBs of the luma component.

4.2 *Prediction Mode and Prediction Block Structure*

In P and B slices, a `cu_skip_flag` is sent for each CU to indicate whether all associated CBs are coded using skip mode, i.e., by using the so-called merge mode for inter-picture prediction (as explicitly described in Sect. 5) and not sending any residual data. To leverage spatial correlation of neighboring CUs, the context of the `cu_skip_flag` depends on whether the top and left neighboring CUs are also skipped. For every non-skipped CU, a regular coded flag called pred_mode_flag is sent to indicate the prediction mode, i.e., the decision whether the CU is either intra coded or inter coded.[9]

Every non-skipped CU may be further subdivided into PUs, as shown for the example in Fig. 6. The syntax element part_mode indicates if and how each CU is partitioned for the purpose of prediction. The choice of prediction block structures for each CU depends on whether the CU is intra-coded or inter-coded; accordingly, part_mode is binarized and coded differently for intra-coded CUs and inter-coded CUs, as shown in Fig. 7.

Intra-coded CUs can have a single PU (referred to as PART_2Nx2N) equal to the size of the CU, or be subdivided into four smaller PUs (referred to as PART_NxN). PART_NxN, however, is only allowed when the CU size is equal to the minimum allowed CU size. If the CU size is greater than the minimum allowed CU size, split_cu_flag is used instead of part_mode to avoid redundant signaling. For instance, if the minimum CU size is 8×8 in terms of luma samples, a CU of size 16×16 with four 8×8 PUs is signaled with split_cu_flag $= 1$, and part_mode=PART_2Nx2N rather than split_cu_flag$= 0$ and part _mode=PART_NxN. Accordingly, part_mode is not signaled but inferred to be PART_2Nx2N when the CU size is greater than the minimum allowed CU size. If the CU size is equal to the minimum allowed CU size, part_mode is coded using a flag with a fixed context for a given slice type.

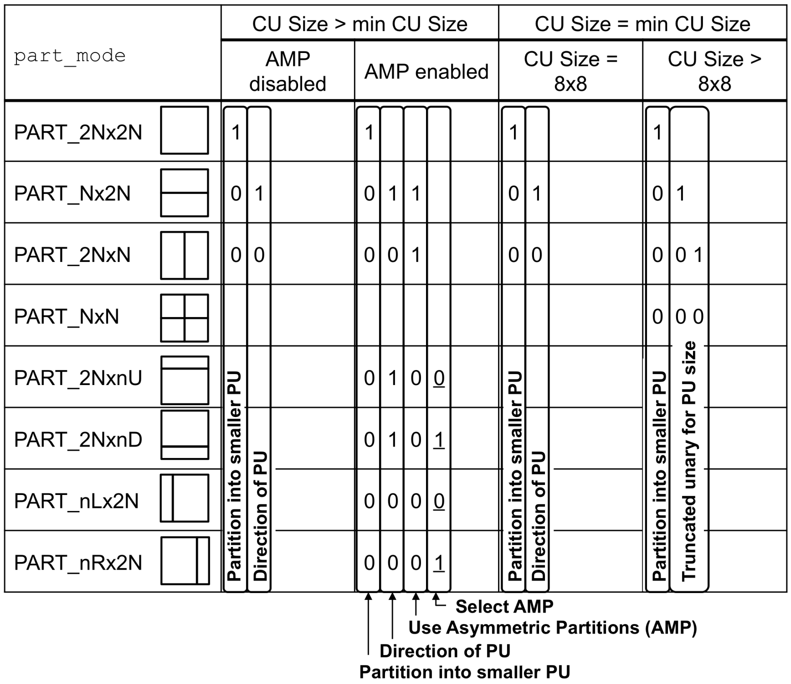
Inter-coded CUs have more prediction partitioning options than intra-coded CUs. In addition to PART_2Nx2N and PART_NxN, inter-coded CU can also be subdivided into two rectangular PUs in either horizontal (PART_Nx2N) or veritical directions (PART_2NxN). In case of enabling the so-called asymmetric motion partitioning (AMP), the additional prediction partitioning possibilities PART_2NxnU, PART_2NxnD, PART_nLx2N, and PART_nRx2N are supported. Custom binarization is used for part_mode as shown in Fig. 7b. The first bin indicates whether or not the CU is partitioned into smaller PUs. If the CU size is greater than the minimum allowed CU size, the second bin indicates the direction of the partition (vertical or horizontal), the third bin indicates whether AMP is used and if so, then a fourth bin is sent to indicate which asymmetric partition is used in the given direction. If the CU size is equal to the minimum allowed CU size, AMP is not allowed and truncated unary coding is used to indicate if the partitions are PART_Nx2N, PART_2NxN, and PART_NxN, respectively.[10] A different context is used for the first and second bin to estimate the probabilities of whether the PU is partitioned into smaller PUs, and the direction of the PU. Two different contexts are used for the third bin depending on when the CU size is greater or equal to the minimum allowed CU size. In the former, the context is based on the probability of whether asymmetric partitions are used, while in the latter, the context is based on the prob-

[9] pred_mode_flag is not sent for CUs in I slices since they are all intra-coded.

[10] Note that the minimum allowed inter-coded PART_NxN size is 8×8, so for CU size equal to 8×8, the only allowed partitions are PART_2NxN and PART_Nx2N, and cMax of 2 is used for truncated unary.

part_mode	CU Size > min CU Size	CU Size = min CU Size
PART_2Nx2N □	Inferred	1
PART_NxN ⊞	Not Allowed	0

(a) Intra-coded CU



part_mode	CU Size > min CU Size		CU Size = min CU Size	
	AMP disabled	AMP enabled	CU Size = 8x8	CU Size > 8x8
PART_2Nx2N	1	1	1	1
PART_Nx2N	0 1	0 1 1	0 1	0 1
PART_2NxN	0 0	0 0 1	0 0	0 0 1
PART_NxN				0 0 0
PART_2NxnU		0 1 0 $\underline{0}$		
PART_2NxnD		0 1 0 $\underline{1}$		
PART_nLx2N		0 0 0 $\underline{0}$		
PART_nRx2N		0 0 0 $\underline{1}$		

Partition into smaller PU
Direction of PU
Partition into smaller PU
Direction of PU
Partition into smaller PU
Truncated unary for PU size
Select AMP
Use Asymmetric Partitions (AMP)
Direction of PU
Partition into smaller PU

(b) Inter-coded CU

Fig. 7: Context selection and binarization of part_mode. Underlined symbols are bypass coded.

ability of whether PART_NxN is used. To reduce the number of regular coded bins, the fourth bin (for AMP) is bypass coded.

4.3 Signaling of Special Coding Modes

HEVC supports two special coding modes, which are invoked on a CU level: the so-called I_PCM mode and the lossless coding mode. Both modes, albeit similar in appearance to some degree, serve different purposes and hence, use different syntax elements for providing different functionalities.

A pcm_flag is sent to indicate whether all samples of the whole CU are coded with *pulse code modulation* (PCM), such that prediction, transform, quantization, and entropy coding as well as their counterparts on the decoder side are simply bypassed. This I_PCM mode, however, is only allowed for intra-coded CUs with prediction partitioning mode PART_2Nx2N.[11] The pcm_flag is coded with the termination mode of the arithmetic coding engine, since in most cases I_PCM mode is not used, and if it is used, the arithmetic coding engine must be flushed and the resulting CABAC codeword must be byte aligned before the PCM sample values can be written directly into the bitstream with fixed length codewords.[12] This procedure also indicates that the I_PCM mode is in particular useful in cases, where the statistics of the residual signal would be such that otherwise, an excessive amount of bits would be generated when applying the regular CABAC residual coding process.

The option of *lossless coding*, where for coding of the prediction residual both the transform and quantization (but not the entropy coding) are bypassed, is also enabled on a CU level and indicated by a regular coded flag called cu_transquant_bypass_flag. The resulting samples of the losslessly represented residual signal in the spatial domain are entropy-coded by the CABAC residual coding process (see Sect. 6), as if they were conventional transform coefficient levels. Note that in lossless coding mode, both in-loop filters are also bypassed in the reconstruction process (which is not necessarily the case for I_PCM), such that a mathematically lossless (local) reconstruction of the input signal is achieved.

4.4 Signaling of Block-based Quantization Parameter Change

In the regular, i.e., lossy residual coding process, a different quantizer step size can be used for each CU to improve bit allocation, rate control, or both. Rather than sending the absolute quantization parameter (QP), the difference in QP steps relative to the slice QP is sent in the form of a so-called delta QP. This functionality can be enabled in the picture parameter set (PPS) by using the syntax element cu_qp_delta_enabled_flag.

In H.264/AVC, mb_qp_delta is used to provide the same instrument of delta QP at the macroblock level. The value of mb_qp_delta can range from $-(26+\text{QpBdOffset}_{\text{Y}}/2)$ to $25+\text{QpBdOffset}_{\text{Y}}/2$. For 8-bit video, this is -26 to 25, while for 10-bit video this is -32 to 31. mb_qp_delta is unary coded and thus requires up to 53 bins for 8-bit video and 65 bins for 10-bit video. All bins are regular coded.

In HEVC, delta QP is represented by the two syntax elements cu_qp_delta_abs and cu_qp_delta_sign_flag, if cu_qp_delta_enabled_flag in the PPS indicates so. The sign is sent separately from the absolute value, which reduces

[11] I_PCM is not allowed for intra-coded 4×4 blocks.

[12] Note that the PCM sample bit depth (i.e. wordlength) for luma and chroma samples can be specified independently in the SPS.

the average number of bins by half [27]. cu_qp_delta_sign_flag is only sent if the absolute value is non-zero. The absolute value is binarized with TrU (cMax=5) as the prefix and EG0 as the suffix [94]. The prefix is regular coded and the suffix is bypass coded. The first bin of the prefix uses a different context than the other four bins in the prefix (which share the same context) to capture the probability of having a zero-valued delta QP. Note that syntax elements for delta QP are only signaled for CUs that have non-vanishing prediction errors (i.e., at least one non-zero transform coefficient). Conceptually, the delta QP is an element of the transform coding part of HEVC and hence, can also be interpreted as a syntax element that is always signaled at the root of the RQT, regardless which transform block partitioning is given by the RQT structure. Tab. 3 shows examples of how delta QP is signaled for H.264/AVC and HEVC.

Table 3: Coding of delta QP in HEVC and H.264/AVC. Underlined symbols are bypass coded.

Value	**HEVC**		**H.264/AVC**
	cu_qp_delta_abs	cu_qp_delta_sign_flag	mb_qp_delta
0	0	n/a	0
1	10	0	10
-1	10	1	110
2	110	0	1110
-2	110	1	1111 0
3	1110	0	1111 10
-3	1110	1	1111 110
4	11110	0	1111 1110
-4	11110	1	1111 1111 0
5	11111 0	0	1111 1111 10
-5	11111 0	1	1111 1111 110
6	11111 100	0	1111 1111 1110
-6	11111 100	1	1111 1111 11110
25	11111 111100101	0	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 10
-26	11111 111100110	1	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 0

4.5 Signaling of SAO Parameters

SAO is a form of in-loop filtering that was introduced in HEVC. It is used to process the output of samples from the deblocking filter process and is the last step of the decoding process. SAO involves sample based processing rather than block based processing. There are two types of filtering: edge offset and band offset.

Edge offset (EO) involves comparing the sample and its neighboring sample values in one of four angular directions (horizontal, vertical, 45°, 135°).[13] The sample is compared to its neighbors in the selected direction (e.g., the sample has a lower value than both its neighbors); based on the comparison, the sample is assigned to a category, which determines the offset that is added to the sample. The value of the offset for a given category is set by the encoder. Band offset (BO) involves dividing the intensity range into four bands and then adding a different offset to the sample depending which band its sample intensity belongs to. For more details on SAO, please refer to Chap. "In-Loop Filters in HEVC".

The type, direction and offsets used to define the SAO filter can change for each CTB; however, all samples belonging to a CTB are processed with the same SAO filter (but luma and chroma CTBs may use different SAO filters). The SAO type is signaled using `sao_type_idx_luma` and `sao_type_idx_chroma` with TrU binarization. The first bin indicates whether the SAO filter is enabled and is regular coded, while the second bin indicates if edge or band offset is used and is bypass coded.

If edge offset is used, the direction of the edge is signaled using `sao_eo_class_luma` and `sao_eo_class_chroma` with FL binarization of 2 bins, all of which are bypass coded. If band offset is used, four `sao_band_position` syntax elements are signaled to indicate the start position of each band with a FL binarization of 5 bins, all of which are bypass coded.

For both types of SAO filtering, four `sao_offset_abs` are signaled (one for each category or band) using TrU with cMax computed by Eq. 4 and all bins are bypass coded.

$$\text{cMax} = (1 << (\min(\text{bitDepth}, 10) - 5)) - 1 \tag{4}$$

For the band offset, the `sao_offset_sign` is signaled only when the offset is non-zero to reduce the total number of bins [38], while for edge offset the sign is inferred from the category [42].

To leverage the spatial correlation across CTBs, `sao_merge_left_flag` and `sao_merge_up_flag` are used to indicate if SAO parameters can be inherited from neighboring CTBs, which reduces signaling overhead. Both of these flags are regular coded using separate context models.

Significant effort was made to reduce the number of regular coded bins required to represent SAO filter syntax elements. As a result, the only regular coded bins are the merge flags and the first bin of the `sao_type_idx_luma` and `sao_type_idx_chroma` with the latter indicating whether SAO is enabled for luma and chroma CTBs, respectively.

[13] Direction is also referred to as class in the HEVC specification.

4.6 Comparison of HEVC and H.264/AVC

Tab. 4 highlights the differences in signaling between the CTU/CU layer in HEVC and the macroblock (MB) layer in H.264/AVC, when processing 8-bit video. For a comparable block partitioning, HEVC typically produces fewer regular coded bins than H.264/AVC. At the same time, some of those regular coded bins in addition to those of the skip flag are adaptively selected based on CU depth, size and neighbors in HEVC, which improves coding efficiency relative to H.264/AVC. In general, however, the total amount of bits spent for signaling at the CTU/CU or MB layer is lower by more than an order of magnitude compared to the total amount of bits spent for transform coefficient level coding. As already discussed above and summarized in Tab. 4, the majority of bypass bins for the SAO parameters are due to the signaling of the offsets, while for H.264/AVC an excessive number of bins is only generated in the rare cases where large delta QP values have to be transmitted.

Table 4: Differences in signaling between CTU/CU layer in HEVC and MB layer in H.264/AVC.

	HEVC	**H.264/AVC**
Prediction and Coding Block Structure and Prediction Mode	`cu_skip_flag`, `split_cu_flag`, `pred_mode_flag`, `part_mode`	`mb_skip_flag`, `mb_type`, `sub_mb_type`
Maximum number of bins for delta QP	5 (regular), 10 (bypass)	53 (regular)
Maximum number of bins for SAO parameters	4 (regular), 113 (bypass)	n/a

5 Prediction Unit Syntax Elements

The prediction unit (PU) syntax elements describe how the prediction is performed in order to reconstruct the samples belonging to each PU. Coding efficiency improvements have been made in HEVC for both modeling and coding of motion parameters and intra prediction modes. While H.264/AVC uses a *single* motion vector predictor (unless direct mode is used) and a *single* most probable mode (MPM), HEVC uses *multiple* candidate predictors or MPMs together with an index or flag for signaling the selected predictor or MPM, respectively. In addition, HEVC provides a mechanism for exploiting spatial and temporal dependencies with regard to motion modeling by *merging* neighboring blocks with identical motion parameters. This has been found to be particularly useful in combination with quadtree-based block partitioning, since a pure hierarchical subdivision approach may lead to partitionings with suboptimal rate-distortion behavior [102, 46, 35]. Also, due to the significant increased number of angular intra prediction modes relative to H.264/AVC, three MPMs for each PU are considered in HEVC.

This section will discuss how the various PU syntax elements are processed in terms of binarization, context modeling, and context assignment. Also, aspects related to parsing dependencies and throughput for the various prediction parameters are considered.

5.1 Motion Data Coding

In HEVC, motion data can be either signaled using merge mode or directly using motion vectors differences, reference indices, and inter-prediction direction.

5.1.1 Signaling of merge mode

In HEVC, merge mode enables motion data (i.e., prediction direction, reference index and motion vectors) to be inherited from a spatial or temporal (co-located) neighbor. A list of merge candidates are generated from these neighbors. `merge_flag` is signaled to indicate whether merge is used in a given PU. If merge is used, then `merge_idx` is signaled to indicate from which candidate the motion data should be inherited. `merge_idx` is coded with truncated unary, which means that the bins are parsed until a zero bin value is reached or when the number of bins is equal to the cMax, the max allowed number of bins.

Determining how to set cMax involved evaluating the throughput and coding efficiency trade-offs in a core experiment [12]. For optimal coding efficiency, cMax should be set to equal the merge candidate list size of the PU. Furthermore, `merge_flag` should not be signaled if the list is empty. However, this makes parsing depend on list construction, which is needed to determine the list size. Constructing the list requires a large amount of computation since it involves reading from multiple locations (i.e., fetching the co-located neighbor and spatial neighbors) and performing several comparisons to prune the list; thus, dependency on list construction would significantly degrade parsing throughput [108, 36].

To decouple the list generation process from the parsing process such that they can operate in parallel in HEVC, cMax is signaled in the slice header using `five_minus_max_num_merge_cand` and does not depend on list size. To compensate for the coding loss due to the fixed cMax, combined bi-predictive and zero motion vector candidates are added when the list size is less than the maximum number of allowed candidates as defined by cMax [79]. This also ensures that the list is never empty and that `merge_flag` is always signaled [107]. For more details on candidate list construction please refer to Chap. "Inter-Picture Prediction in HEVC".

5.1.2 Signaling of motion vector differences, reference indices, and inter-prediction direction

If merge mode is not used, then the motion vector is predicted from its neighboring blocks and the difference between motion vector (mv) and motion vector prediction (mvp), referred to as *motion vector difference* (mvd), is signaled:

$$\text{mvd}(x,y) = \text{mv}(x,y) - \text{mvp}(x,y)$$

In H.264/AVC, a single predictor is calculated for mvp from the median of the left, top and top-right spatial 4×4 neighbors.

In HEVC, advanced motion vector prediction (AMVP) is used, where several candidates for mvp are determined from spatial and temporal neighbors [40]. A list of mvp candidates is generated from these neighbors, and the list is pruned to remove redundant candidates such that there is a maximum of 2 candidates. A syntax element called `mvp_l0_flag` (or `mvp_l1_flag` depending on the reference list) is used to indicate which candidate is used from the list as the mvp. To ensure that parsing is independent of list construction, `mvp_l0_flag` is signaled even if there is only one candidate in the list. The list is never empty as the zero motion vector is used as the default candidate.

In HEVC, improvements were also made on the coding process of mvd itself. In H.264/AVC, the first 9 bins of mvd are regular coded truncated unary bins, followed by bypass coded 3rd order Exp-Golomb bins. In HEVC, the number of regular coded bins for mvd is significantly reduced [60]. Only the first two bins are regular coded (`abs_mvd_greater0_flag`, `abs_mvd_greater1_flag`), followed by bypass coded first-order Exp-Golomb (EG1) bins (`abs_mvd_minus2`).

In H.264/AVC, context selection for the first bin in mvd depends on whether the sum of the motion vectors of the top and left 4×4 neighbors are greater than 32 (or less than 3). This requires 5-bit storage per neighboring motion vector, which accounts 24,576 of the 30,720-bit CABAC line buffer needed to support a 4k×2k sequence. [95] highlighted the need to reduce the line buffer size in HEVC by modifying the context selection logic. Accordingly, all dependencies on the neighbors were removed and the context is selected based on the binIdx (i.e., whether it is the first or second bin) [96, 83].

To maximize the impact of fast bypass coding, the bypass coded bins for both the horizontal (x) and vertical (y) components of mvd are grouped together in HEVC [67]. For instance, for a motion vector difference of $\text{mvd}(x,y) = (2,2)$, the coding order is $1111\underline{0000}$, where underlined values are bypass coded. Without bypass grouping, the coding order is $11\underline{00}11\underline{00}$. If 4 bypass bins can be processed in a single cycle, enabling bypass grouping reduces the number of cycles required to process the motion vector by one.

In HEVC, reference indices `ref_idx_l0` and `ref_idx_l1` are coded with truncated unary regular coded bins, which is the same as for H.264/AVC; the maximum length of the truncated unary binarization, cMax, is dictated by the reference picture list size. However, in HEVC only the first two bins are regular coded [71], whereas all bins are regular coded in H.264/AVC. In both HEVC and H.264/AVC, the regular coded bins of the reference indices for different reference picture lists share the same set of contexts. The inter-prediction direction (list 0, list 1 or bi-directional) is signaled using (inter_pred_idc) with custom binarization.

5.2 Intra Prediction Mode Coding

Similar to motion data coding, a most probable mode (MPM) is calculated for intra mode coding. In H.264/AVC, the minimum mode of the top and left neighbors is used as MPM. `prev_intra4x4_pred_mode_flag` (or `prev_intra8x8_pred_mode_flag`) is signaled to indicate whether the most probable mode is used. If the MPM is not used, the remainder mode `rem_intra4x4_pred_mode_flag` (or `rem_intra8x8_pred_mode_flag`) is signaled.

In HEVC, additional MPMs are used to improve coding efficiency. A candidate list of most probable modes with a fixed length of three is constructed based on the left and top neighbors. The additional candidate modes (DC, planar, vertical) can be added if the left and top neighbors are the same or unavailable. Note that the top neighbors outside current CTU are considered unavailable in order to avoid the need for a line buffer.[14] The prediction flag (`prev_intra_pred_mode_flag`) is signaled to indicate whether one of the most probable modes is used. If an MPM is used, a most probable mode index (`mpm_idx`) is signaled to indicate which candidate to use. It should be noted that in HEVC, the order in which the coefficients of the residual are parsed (e.g., diagonal, vertical or horizontal) depends on the reconstructed intra mode (i.e., the parsing of the TU data that follows depends on list construction and intra mode reconstruction). Thus, the candidate list size was limited to three for reduced computation to ensure that it would not affect entropy decoding throughput [84, 26].

The number of regular coded bins was reduced for intra mode coding in HEVC relative to the corresponding part in H.264/AVC, where both the flag and the 3 fixed-length bins of the remainder mode are regular coded using two separate context models. In HEVC, the flag is regular coded as well, but the remainder mode is a fixed-length 5-bin value that is entirely bypass coded. The most probable mode index (`mpm_idx`) is also entirely bypass coded. The number of contexts used to code `intra_chroma_pred_mode` is reduced from 4 to 1 for HEVC relative to H.264/AVC. To maximize the impact of fast bypass coding, the bypass coded bins for luma intra prediction mode coding within a CU are grouped together in HEVC [23]. This is beneficial when the partition mode is PART_NxN, and there are four sets of prediction modes.

[14] For more details on MPM list construction please refer to Chap. “Intra-Picture Prediction in HEVC”.

Table 5: Differences between prediction unit coding in HEVC and H.264/AVC.

Properties	**HEVC**			**H.264/AVC**	
	Intra Mode	**AMVP**	**Merge**	**Intra Mode**	**MVP**
Max number of candidates in list	3	2	5	1	1
Spatial neighbor	used	used	used	used	used
Temporal co-located neighbor	not used	used	used	not used	not used
Number of contexts	2	10	2	6	20
Max regular coded bins per PU	2	16	2	7	98

5.3 Comparison of HEVC and H.264/AVC

The differences between H.264/AVC and HEVC in signaling of syntax elements at the PU layer are summarized in Tab. 5. HEVC uses both spatial and temporal neighbors as predictors, while H.264/AVC only uses spatial neighbors (unless direct mode is enabled). In terms of the impact of the throughput improvement techniques, HEVC has around 6× fewer maximum regular coded bins per inter-predicted PU than H.264/AVC. HEVC also requires around 2× fewer contexts for PU syntax elements than H.264/AVC.

6 Transform Unit Syntax Elements

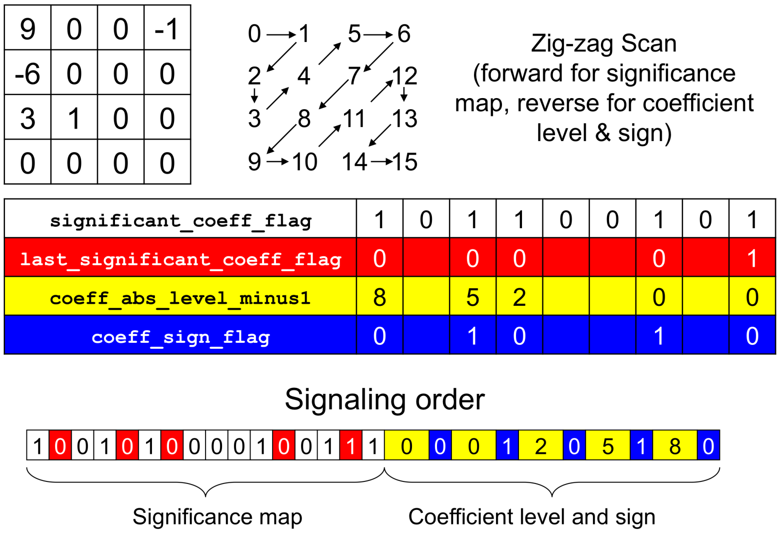
In video coding, both intra and inter prediction are used to reduce the amount of data that needs to be transmitted. In addition, rather than sending the original samples of the prediction signal, an appropriately quantized approximation of the prediction error is transmitted. To this end, the prediction error is blockwise transformed from spatial to frequency domain, thereby decorrelating the residual samples and performing an energy compaction in the sense that, after quantization, the signal can be represented in terms of a few non-vanishing coefficients. The method of signaling the quantized values and frequency positions of these coefficients is referred to as *transform coefficient coding*.

Syntax elements related to transform coefficient coding account for a significant portion of the bin workload as shown in Tab. 6. At the same time, those syntax elements also account for a significant portion of the total number of bits for a compressed video, and as a result the compression of quantized transform coefficients significantly impacts the overall coding efficiency. Thus, transform coefficient coding with CABAC must be carefully designed in order to balance coding efficiency and throughput demands. Accordingly, as part of the HEVC standardization process, a core experiment on coefficient scanning and coding was established to investigate tools related to transform coefficient coding [98].

	HEVC					H.264/AVC		
Common conditions	AI MAIN	LP MAIN	LB MAIN	RA MAIN	worst case	HierB	HierP	worst case
CTU/CU bins	5.4%	15.8%	16.7%	11.7%	1.4%	27.0%	34.0%	0.5%
PU bins	9.2%	20.6%	19.5%	18.8%	5.0%	23.4%	26.3%	15.8%
TU bins	85.4%	63.7%	63.8%	69.4%	94.0%	49.7%	39.7%	83.7%

Table 6: Distribution of bins in CABAC for HEVC and H.264/AVC under common test conditions [4, 11] and for the worst case. Generated bins are discriminated along the HEVC categories CTU/CU, PU, and TU as well as their corresponding counterparts in H.264/AVC.

This section describes how transform coefficient coding evolved from H.264/AVC to the first test model of HEVC (HM1.0) to the Final Draft International Standard (FDIS) of HEVC (HM10.0), and discusses the reasons behind design choices that were made. Many of the throughput improvement techniques were applied, and new tools for improved coding efficiency were simplified. As a reference for the beginning and end points of the development, Fig. 8 and Fig. 9 show examples of transform coefficient coding for 4×4 blocks in H.264/AVC and HEVC, respectively.



9	0	0	-1
-6	0	0	0
3	1	0	0
0	0	0	0

0 1 5 6
2 4 7 12
3 8 11 13
9 10 14 15
Zig-zag Scan
(forward for significance map, reverse for coefficient level & sign)

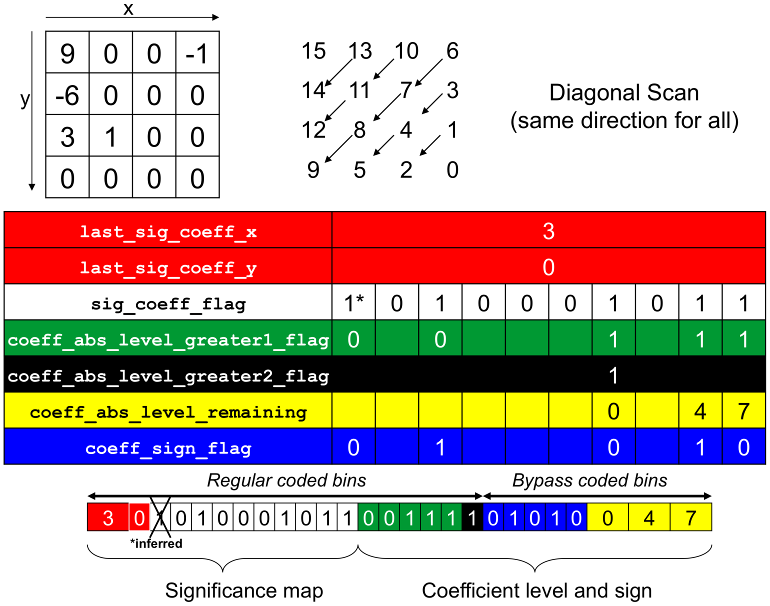
significant_coeff_flag	1	0	1	1	0	0	1	0	1
last_significant_coeff_flag	0		0	0			0		1
coeff_abs_level_minus1	8		5	2			0		0
coeff_sign_flag	0		1	0			1		0

Signaling order

1	0	0	1	0	1	0	0	0	0	1	0	0	1	1	1	0	0	0	1	2	0	5	1	8	0

Significance map
Coefficient level and sign

Fig. 8: Example of CABAC-based transform coefficient coding for a 4×4 transform block in H.264/AVC. Note, however, that the corresponding bins for signaling of the absolute level (in yellow) are not explicitly shown.



9	0	0	-1
-6	0	0	0
3	1	0	0
0	0	0	0

last_sig_coeff_x	3									
last_sig_coeff_y	0									
sig_coeff_flag	1*	0	1	0	0	0	1	0	1	1
coeff_abs_level_greater1_flag	0		0				1		1	1
coeff_abs_level_greater2_flag							1			
coeff_abs_level_remaining							0		4	7
coeff_sign_flag	0		1				0		1	0

Fig. 9: Example of transform coefficient coding for a 4×4 transform block in HEVC. Note, however, that the corresponding bins for signaling of the "last" information (in red) and absolute level remaining (in yellow) are not explicitly shown.

6.1 Transform Block Structure

As already discussed in Sect. 3.1, transform coding in HEVC involves a tree-structured variable block-size approach with supported transform block sizes of 4×4, 8×8, 16×16, and 32×32. This means that the actual transform block sizes, used to code the prediction error of a given CU, can be selected based on the characteristics of the residual signal by using a quadtree-based partitioning, also known as residual quadtree (RQT), as illustrated in Fig. 10. While this larger variety of transform block partitioning relative to H.264/AVC provides significant coding gains, it also has implications in terms of implementation costs, both in terms of memory bandwidth and computational complexity. To address this issue, HEVC allows to restrict the RQT-based transform block partitioning by four parameters, signaled by corresponding syntax elements in the SPS: the maximum and minimum allowed transform block size (in terms of block width) $n_{\max}$ and $n_{\min}$, respectively, and the maximum depth of the RQT $d_{\max}$, with the latter given both individually for intra-picture and inter-picture prediction. Note, however, that there is a rather involved interdependency between these parameters (and other syntax elements), such that, e.g., implicit subdivisions or implicit leaf nodes of the RQT may occur. For more details, please refer to Chap. "Block Structures and Parallelism Features in HEVC".

The signaling of the transform block structure for each CU is similar to that of the coding block structure at the CTU level. For each node of the RQT, a flag called `split_transform_flag` is signaled to indicate whether a given transform block should be further subdivided into four smaller TBs. Context modeling for the coding of this flag involves three different contexts with its related context increment equal to $5 - \log_2(\text{TrafoSize})$, where TrafoSize denotes the block width of the corresponding luma transform block at the given RQT depth. Note that for the choice of a luma CTB size of 64, $n_{\max} = 32$, $n_{\min} = 4$, and $d_{\max} = 4$, an implicit leaf node is implied for the case of $\text{TrafoSize} = 4$, whereas an implicit subdivision is given for a luma CB size of 64 at RQT depth equal to 0. Tab. 7 and Fig. 11 illustrate an example of this configuration. Therefore, even if up to five different RQT levels are permitted, only up to three different context models are required for coding of `split_transform_flag`. Note that the signaling of `split_transform_flag` at the RQT root is omitted if the quantized residual of the corresponding CU contains no non-zero transform coefficient at all, i.e., if the corresponding coded block flag at the RQT root (see Sect. 6.3) is equal to 0.

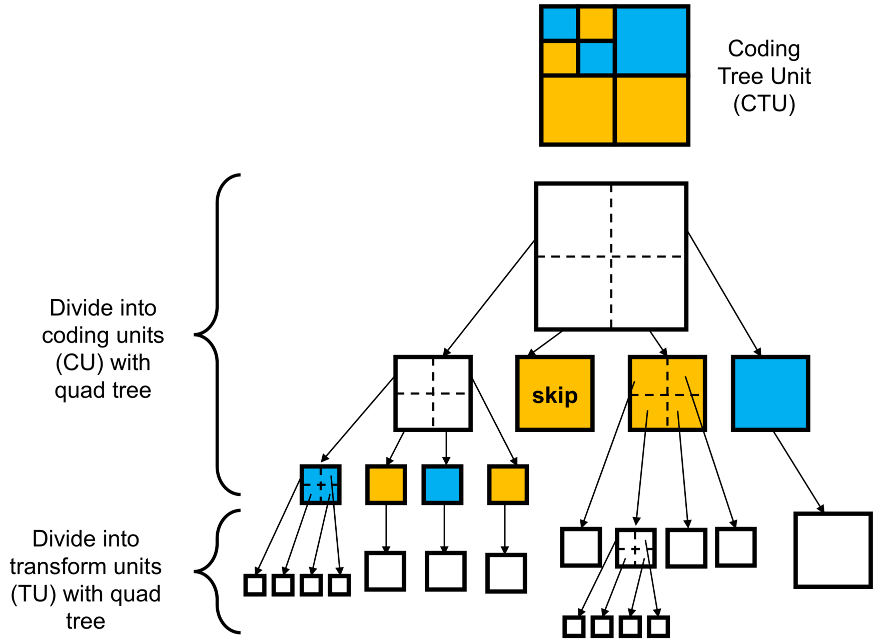


Fig. 10: Illustration of residual quadtrees (one for each CU) used to signal transform units for residual coding of CUs. Note that the same relationships and comments as given in Fig. 6 apply here as well.

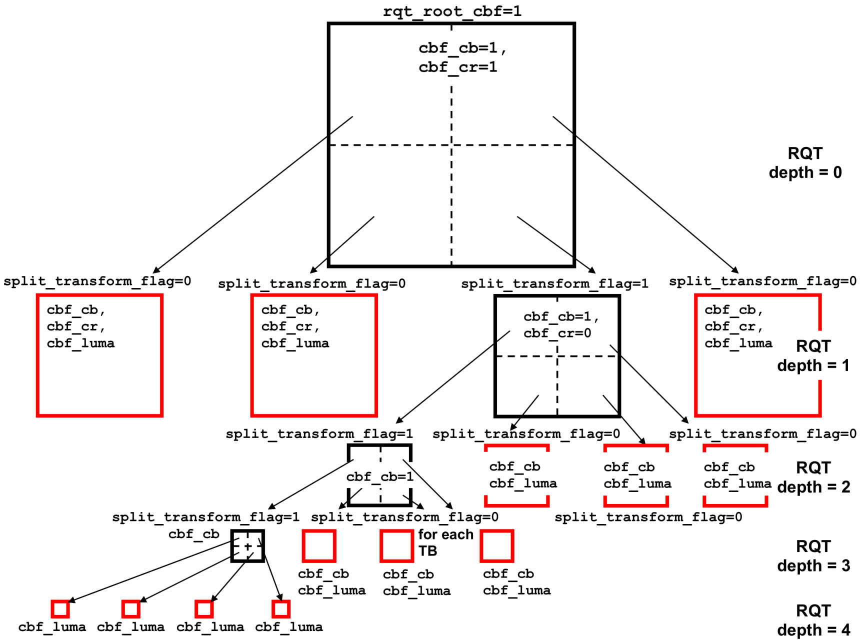


Fig. 11: Illustration of signaling of split_transform_flag, cbf_luma, cbf_cb, and cbf_cr for an RQT with depth 4. Note that at RQT depth = 0, no split_transform_flag is signaled since an implicit transform split occurs for CU of 64 as $n_{\max} = 32$. cbf_luma is only signaled for leaf transform blocks (highlighted in red). cbf_cb and cbf_cr are signaled for the root node and all nodes where the corresponding CBF at the parent node is non-zero, except for the nodes related to TrafoSize $= 4$.

RQT depth	Transform Size	split_transform_flag (ctxInc)	cbf_luma (ctxInc)	cbf_cb, cbf_cr (ctxInc)
0	n/a	n/a	1	0
1	32×32	0	0	1
2	16×16	1	0	2
3	8×8	2	0	3
4	4×4	n/a	0	n/a

Table 7: Derivation of context increment (ctxInc) for split_transform_flag, cbf_luma, cbf_cb, and cbf_cr for the example in Fig. 11.

6.2 Transform Skip

For regions or blocks with many sharp edges (e.g., as typically given in screen content coding), coding gains can be achieved by skipping the transform [44, 63]. When the transform is skipped for a given block, the prediction error in the spatial domain is quantized and coded in the same manner as for transform coefficient coding (i.e., the quantized block samples of the spatial error are coded as if they were quantized transform coefficients). The so-called *transform skip* mode is only allowed for 4×4 TUs and only if the corresponding functionality is enabled by the `transform_skip_enabled_flag` in the PPS. Signaling of this mode is performed by using the `transform_skip_flag`, which is coded using a single fixed context model.

6.3 Coded Block Flags

At the top level of the hierarchy of significance flags, as already explained in Sect. 3.1, coded block flags (CBFs) are signaled for the RQT root, i.e., at the CU level in form of the `rqt_root_cbf` and for subsequent luma and chroma TBs in the form of `cbf_luma` and `cbf_cb`, `cbf_cr`, respectively. `rqt_root_cbf` is only coded and transmitted for inter-predicted CUs that are not coded in merge mode using a single PU (PART_2Nx2N)[15]; for that a single context model is used. While signaling of `cbf_luma` is only performed at the leaf nodes of the RQT, provided that a non-zero `rqt_root_cbf` was signaled before, the chroma CBFs `cbf_cb` and `cbf_cr` are also transmitted at each internal node as long as a corresponding non-zero chroma CBF at its parent node occurred. For coding of both `cbf_cb` and `cbf_cr`, four contexts are used such that the corresponding context increment depends on the RQT depth (with admissible values between 0 and 3, since for the case of TrafoSize $= 4$ no chroma CBFs are transmitted), whereas for `cbf_luma` only two contexts are provided with its discriminating context increment depending on the condition RQT depth $= 0$. For more background on the use of RQT and related syntax elements, please refer to Chap. "Block Structures and Parallelism Features in HEVC".

6.4 Significance Map

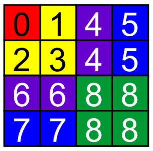
In H.264/AVC, the significance map for each transform block is signaled by transmitting a `significant_coeff_flag` (SIG) for each position to indicate whether the coefficient is non-zero. The positions are processed in an order based on a zig-zag scan. After each non-zero SIG, an additional flag called `last_significant_coeff_flag` (LAST) is immediately sent to indicate whether it is the last non-

[15] Intra-predicted CUs typically have nonzero residual, so `rqt_root_cbf` is not used.

zero SIG; this prevents unnecessary SIG from being signaled. Different contexts are used depending on the position within the 4×4 and 8×8 transform blocks, and whether the bin represents an SIG or LAST. Since SIG and LAST are interleaved, the context selection of the current bin depends on the immediate preceding bin. The dependency of LAST on SIG results in a strong bin to bin dependency for context selection of significance map entries in H.264/AVC as illustrated in Fig. 3.

6.4.1 `sig_coeff_flag` (SIG)

While in HEVC position based context assignment for coding of `sig_coeff_flag` (SIG) is used for 4×4 TBs as shown in Fig. 12, new forms of context assignment for larger transforms were needed. In HM1.0, additional dependencies were introduced in the context selection of SIG for 16×16 and 32×32 TBs to improve coding efficiency. Specifically, the context selection of SIG was calculated based on a local template using 10 (already decoded) SIG neighbors as shown in Fig. 13a [102, 59]. By using this template-based context selection bitrate savings of 1.4 –2.8% were reported [59].

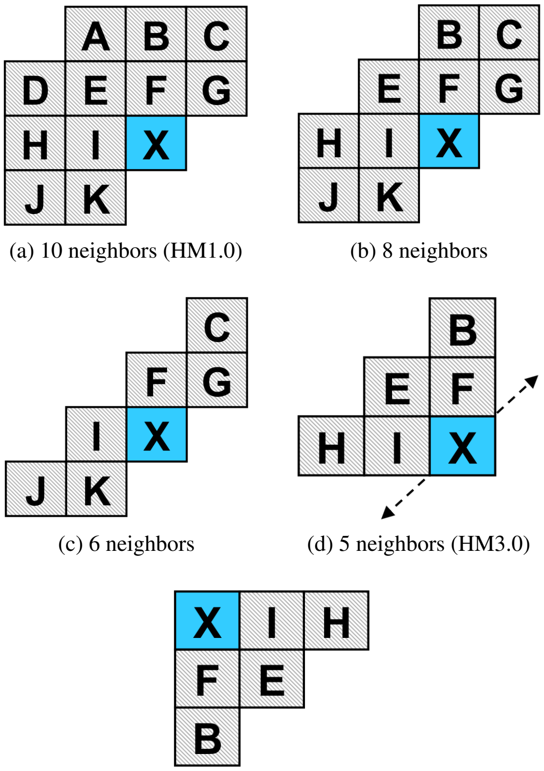


0	1	4	5
2	3	4	5
6	6	8	8
7	7	8	8

Fig. 12: Context index assignment for `sig_coeff_flag` in 4×4 TBs.

To reduce context selection dependencies and storage costs, [86] proposed using fewer neighbors and showed that this could be done without severely sacrificing coding efficiency. For instance, using only a maximum of 8 neighbors (removing neighbors A and D as shown in Fig. 13b) had negligible impact on coding efficiency, while using only 6 neighbors (removing neighbors A, B, D, E and H as shown in Fig. 13c) results in a coding efficiency loss of only 0.2%. This was further extended in [22] for HM2.0, where only a maximum of 5 neighbors was used by removing dependencies on positions G and K, as shown in Fig. 13d. In HM2.0, the significance map was scanned in zig-zag order, so removing the diagonal neighbors G and K is important since those neighbors pertain to the most recently decoded SIG.

Despite reducing the number of SIG neighbors in HM2.0, dependency on the most recently processed SIG neighbors still existed for the positions at the edge of the transform block as shown in Fig. 14a. The horizontal or vertical shift that is required to go from one diagonal to the next in the zig-zag scan causes the previously decoded bin to be one of the neighbors (F or I) that is needed for context selection. In order to address this issue, in HM4.0, a diagonal scan was introduced to replace the zig-zag scan [87] as shown in Fig. 14b. Changing from zig-zag to diagonal



(a) 10 neighbors (HM1.0)

(b) 8 neighbors

(c) 6 neighbors

(d) 5 neighbors (HM3.0)

(e) Inverted for reverse scan (HM4.0)

Fig. 13: Local templates for SIG context selection. X (in blue) represents the current position of the bin being processed.

scan had negligible impact on coding efficiency, but removed the dependency on recently processed SIG for all positions in the TB. In HM4.0, the scan was also reversed (from high frequency to low frequency) [74]. Accordingly, the neighbor dependencies were inverted from top-left to bottom-right, as shown in Fig. 13e.

Dependencies in context selection of SIG for 16×16 and 32×32 TBs were further reduced in HM7.0, where 16×16 and 32×32 TBs are divided into 4×4 subblocks. This will be described in more detail in Sect. 6.4.3 on `coded_sub_block_flag` (CSBF). In HM8.0, 8×8 TBs were also divided into 4×4 subblocks such that all TB sizes above 4×4 are based on a 4×4 subblock processing for a harmonized design [77].

The 8×8, 16×16 and 32×32 TBs are divided into three regions based on frequency, as shown in Fig. 15. The DC, low-frequency and mid/high-frequency regions all use different sets of contexts. To reduce memory size, the contexts for coding the SIG of 16×16 and 32×32 TBs are shared [100, 81].

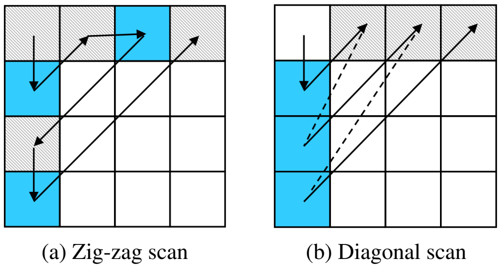


Fig. 14: Scans used to process SIG. Diagonal scan avoids dependency on the most recently processed bin. Context selection for blue positions is affected by values of the neighboring grey positions.

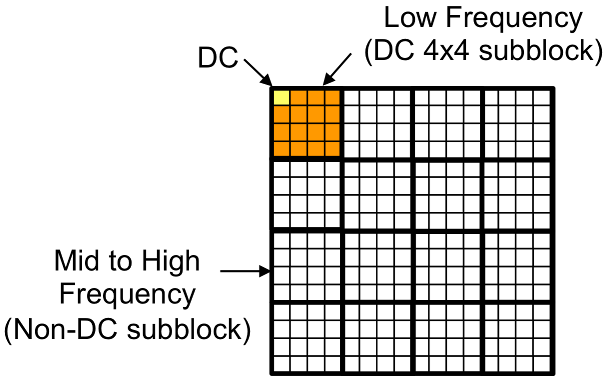


Fig. 15: Regions in 8×8, 16×16 and 32×32 TBs map to different context sets for SIG.

For improved coding efficiency for intra predicted CUs, so-called mode dependent coefficient scanning (MDCS) is used to select between vertical, horizontal, and diagonal scans based on the chosen intra prediction mode [106], as illustrated in Fig. 16. Tab. 8 shows how the scans are assigned based on intra prediction mode, TB size, and component. As mentioned in Sect. 5.2, this requires the intra mode to be decoded before decoding the corresponding transform coefficients. MDCS is only used for 4×4 and 8×8 TBs and provides coding gains of up to 1.2%. Note that for TBs larger than 8×8 and for TBs of inter predicted CUs only the diagonal scan is used.

6.4.2 Last position coding

As mentioned earlier, there are strong data dependencies between `significant_coeff_flag` (SIG) and `last_significant_coeff_flag` (LAST) in H.264/AVC

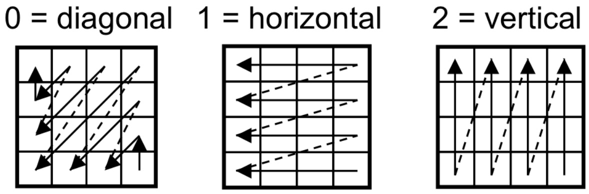


Fig. 16: Diagonal, vertical, and horizontal scans for 4×4 TBs.

Table 8: Mode dependent coefficient scanning: Mapping of intra prediction mode to scans (0 = Diagonal, 1 = Horizontal, 2 = Vertical) for different TB sizes and components.

Intra Prediction Mode	**0 (Planar)**	**1 (DC)**	**2**	**3**	**4**	**5**	**6 to 14**	**15 to 21**	**22 to 30**	**31 to 34**
8×8 (luma)	0					0	2	0	1	0
4×4 (luma or chroma) TB	0					0	2	0	1	0
Otherwise	0									

due to the fact that they are interleaved. [15] proposed grouping several SIG together by transmitting a LAST only once per N number of SIG. If all of the N SIG are zero, LAST is not transmitted. [73] avoids interleaving of SIG and LAST altogether. Specifically, the horizontal (x) and vertical (y) position of the last non-zero SIG in a TB is sent in advance rather than LAST by using the syntax elements `last_sig_coeff_x` and `last_sig_coeff_y`, respectively. For instance, in the example shown in Fig. 9, `last_sig_coeff_x` equal to 3 and `last_sig_coeff_y` equal to 0 are sent before processing the TB rather than signaling LAST for each SIG with value of 1. Signaling the (x, y) position of the last non-zero SIG for each TB was adopted into HM3.0. Note that the SIG in the last scan position is inferred to be 1.

The last position, given by its coordinates in both x and y direction, is composed of a prefix and suffix as shown in Tab. 9. The prefixes `last_sig_coeff_x_prefix` and `last_sig_coeff_y_prefix` are both regular coded using TrU binarization with $\text{cMax} = 2 \cdot \log_2 \text{TrafoSize} - 1$ [70]. A suffix is present when the corresponding prefix is composed of more than 4 bins. In that case, the suffixes `last_sig_coeff_x_suffix` and `last_sig_coeff_y_suffix` are bypass coded using FL binarization. Some of the contexts are shared across the chroma TB sizes to reduce context memory, as shown in Tab. 10. To maximize the impact of fast bypass coding, the bypass coded bins (i.e., the suffix bins) for both the x and y coordinate of the last position are grouped together for each TB in HEVC.

Table 9: Binarization of coordinate values of the last position. Bins belonging to the bypass coded suffixes are underlined.

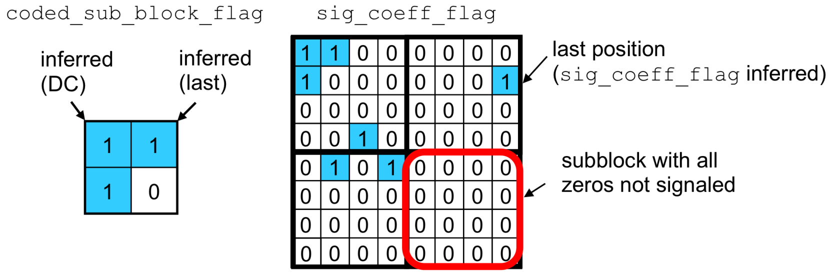
Coordinate Value	**TB** $= 4 \times 4$	**TB** $= 8 \times 8$	**TB** $= 16 \times 16$	**TB** $= 32 \times 32$
0	0	0	0	0
1	10	10	10	10
2	110	110	110	110
3	111	1110	1110	1110
4	n/a	11110 $\underline{0}$	11110 $\underline{0}$	11110 $\underline{0}$
5	n/a	11110 $\underline{1}$	11110 $\underline{1}$	11110 $\underline{1}$
6	n/a	11111 $\underline{0}$	111110 $\underline{0}$	111110 $\underline{0}$
7	n/a	11111 $\underline{1}$	111110 $\underline{1}$	111110 $\underline{1}$
8	n/a	n/a	1111110 $\underline{00}$	1111110 $\underline{00}$
9	n/a	n/a	1111110 $\underline{01}$	1111110 $\underline{01}$
10	n/a	n/a	1111110 $\underline{10}$	1111110 $\underline{10}$
11	n/a	n/a	1111110 $\underline{11}$	1111110 $\underline{11}$
12-15	n/a	n/a	1111111 $\underline{xx}$	11111110 $\underline{xx}$
16-23	n/a	n/a	n/a	111111110 $\underline{xxx}$
24-31	n/a	n/a	n/a	111111111 $\underline{xxx}$

Table 10: Context selection for regular coded prefix bins of the coordinates of the last position `last_sig_coeff_x_prefix` and `last_sig_coeff_y_prefix`.

Bin Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
4x4 luma TB	0	1	2						
8x8 luma TB	3	3	4	4	5				
16x16 luma TB	6	6	7	7	8	8	9		
32x32 luma TB	10	10	11	11	12	12	13	13	14
4x4 chroma TB	15	16	17						
8x8 chroma TB	15	15	16	16	17				
16x16 chroma TB	15	15	15	15	16	16	16		

6.4.3 `coded_sub_block_flag` (CSBF)

As already explained in Sect. 3.1, the number of bins to be transmitted for signaling the significance map is considerably reduced by using a hierarchical signaling scheme of significance flags. Part of this hierarchy is the `coded_sub_block_flag` (CSBF) that indicates for each 4×4 subblock of a TB whether there are non-zero coefficients in the subblock [57][56]. If CSBF is equal to 1, the subblock contains at least one non-zero transform coefficient level and, consequently, SIGs within the subblock are signaled. No SIGs are signaled for a 4×4 subblock that contains all vanishing transform coefficients, since this information is signaled by a CSBF equal to 0. For large TB sizes, a reduction in SIG bins of up to a 30% can be achieved by the use of CSBFs, which corresponds to an overall bin reduction of 3–4% under common test conditions. To avoid signaling of redundant information, the CSBF for the subblocks containing the DC and the last position are inferred to be equal to 1. Fig. 17 shows an example of the hierarchical signaling of an 8×8 significance map.


coded_sub_block_flag
sig_coeff_flag
inferred (DC)
inferred (last)
last position (sig_coeff_flag inferred)
subblock with all zeros not signaled

1	1
1	0

1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 17: Example of the hierarchical signaling of an 8×8 significance map.

In HM7.0, the CSBF was additionally used to further reduce dependencies in the context selection of SIG for 16×16 and 32×32 TBs. Specifically, the neighboring subblocks and their corresponding CSBFs (Fig. 18) are used for context selection rather than the individual SIG neighbors, as shown in Fig. 13e [43]. This context selection scheme was extended to 8×8 TBs in HM8.0 [77]. According to this scheme, the CSBF of the neighboring right and bottom subblocks (CSBF_{right}, CSBF_{bottom}) are used to select one of four patterns shown in Fig. 19: (0,0) maps to pattern 1, (1,0) to pattern 2, (0,1) to pattern 3 and (1,1) to pattern 4. The pattern maps each position within the 4×4 subblock to one of three contexts. As a result, there are no intrinsic dependencies for context selection of SIG within each 4×4 subblock.

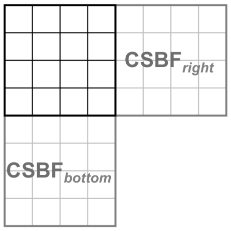
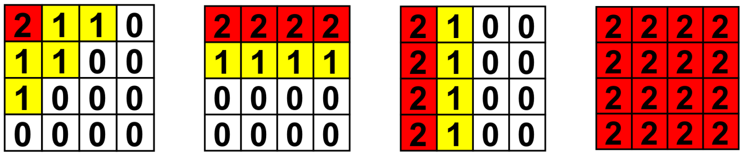


Fig. 18: Neighboring CSBFs (right, bottom) used for SIG context selection.

Reverse diagonal scanning order is used within the subblocks and for the processing order of the subblocks themselves, as shown in Fig. 20 [76]. Both significance map and coefficient levels are processed in this order. As an exception to this rule, for 4×4 and 8×8 TBs to which MDCS is applied, reverse vertical and horizontal scanning orders are used within the subblocks as well as for the processing order of the subblocks themselves. Furthermore, as shown in Tab. 11, different sets of contexts for coding of SIG are used for diagonal and non-diagonal (vertical and horizontal) scans in both 4×4 luma and chroma TBs, and 8×8 luma TBs [77].



2	1	1	0
1	1	0	0
1	0	0	0
0	0	0	0

(a) Pattern 1

2	2	2	2
1	1	1	1
0	0	0	0
0	0	0	0

(b) Pattern 2

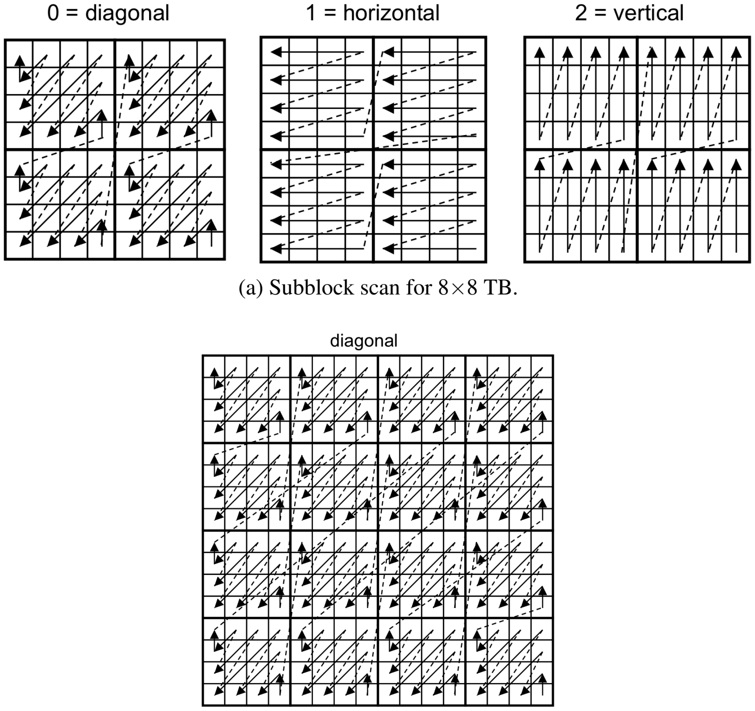
2	1	0	0
2	1	0	0
2	1	0	0
2	1	0	0

(c) Pattern 3

2	2	2	2
2	2	2	2
2	2	2	2
2	2	2	2

(d) Pattern 4

Fig. 19: 4×4 position based mapping for SIG context selection based on CSBF of neighboring subblocks.



(a) Subblock scan for 8×8 TB.

(b) Subblock scan for 16×16 TB. Scan for 32x32 TB is also all diagonal.

Fig. 20: Subblock scans. Scan for 4×4 TB shown in Fig. 16.

6.4.4 Summary of significance map coding in HEVC

Fig. 21 summarize the steps required to code the significance map. This process is repeated for every non-zero TB in HEVC. Tab. 11 summarizes the multiple steps of classification used to assign the 42 contexts of sig_coeff_flag. Contexts 0 to 26 are used for luma coded TBs, while 27 to 41 used for chroma TBs. The contexts are further mapped based on the TB size, the scan direction, whether the subblock is

DC or non-DC, CSBF of neighboring subblocks, and position within the subblock. Note that context 0 is used to code the sig_coeff_flag of the DC position of all luma TBs, and context 27 is used for the DC position of all chroma TBs.

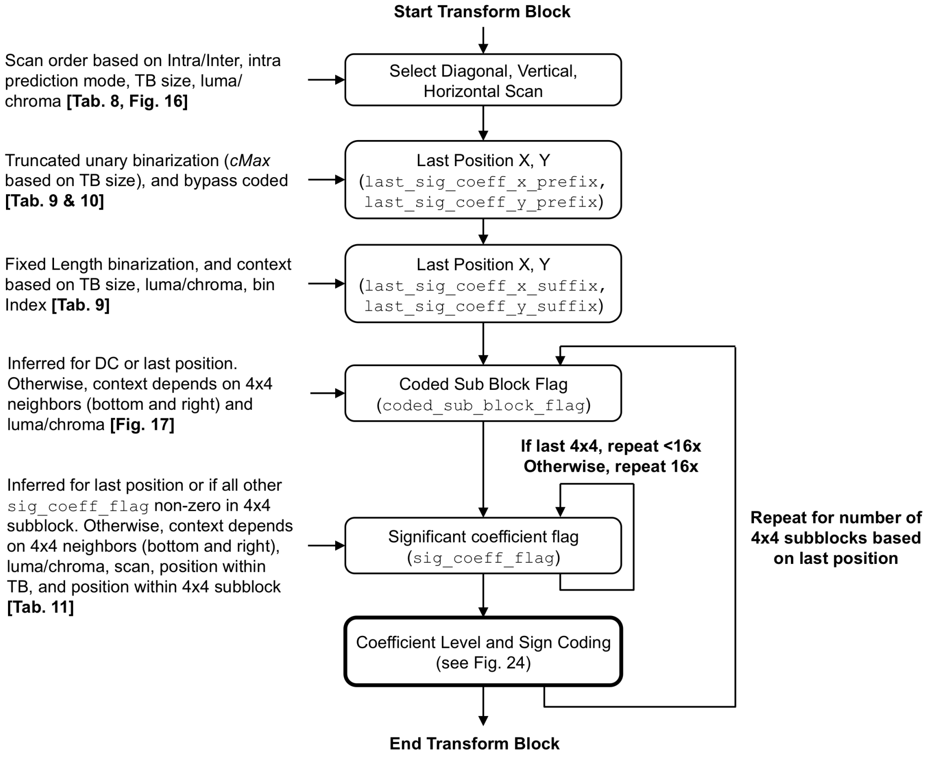

Start Transform Block
Scan order based on Intra/Inter, intra prediction mode, TB size, luma/chroma [Tab. 8, Fig. 16]
Select Diagonal, Vertical, Horizontal Scan
Truncated unary binarization (cMax based on TB size), and bypass coded [Tab. 9 & 10]
Last Position X, Y (last_sig_coeff_x_prefix, last_sig_coeff_y_prefix)
Fixed Length binarization, and context based on TB size, luma/chroma, bin Index [Tab. 9]
Last Position X, Y (last_sig_coeff_x_suffix, last_sig_coeff_y_suffix)
Inferred for DC or last position. Otherwise, context depends on 4x4 neighbors (bottom and right) and luma/chroma [Fig. 17]
Coded Sub Block Flag (coded_sub_block_flag)
If last 4x4, repeat <16x Otherwise, repeat 16x
Inferred for last position or if all other sig_coeff_flag non-zero in 4x4 subblock. Otherwise, context depends on 4x4 neighbors (bottom and right), luma/chroma, scan, position within TB, and position within 4x4 subblock [Tab. 11]
Significant coefficient flag (sig_coeff_flag)
Repeat for number of 4x4 subblocks based on last position
Coefficient Level and Sign Coding (see Fig. 24)
End Transform Block

Fig. 21: Flow chart for coding the syntax elements of a TB in HEVC.

6.5 *Absolute Coefficient Level and Coefficient Sign*

In HEVC, parsing of transform coefficient level information is performed subblock-by-subblock using up to five scan passes for each subblock. The first scan pass is devoted to the SIG flags, as already explained in Sect. 6.4.1 and 6.4.3. In the second and third pass, the two additional flags coeff_abs_level_greater1_flag (ALG1) and coeff_abs_level_greater2_flag (ALG2) are conditionally parsed, indicating for each relevant scan position if the corresponding absolute value of the coefficient level, i.e., the absolute level (AL) is greater than 1 and 2, respectively. However, only up to 8 ALG1 flags and one ALG2 flag are transmitted

Table 11: Context selection of `sig_coeff_flag` based on component, TB size, scan order (Tab. 8), position of subblock within the TB (Fig. 15), and position based context index within 4×4 TB or subblock (SubIdx) (Fig. 12 or Fig. 19, resp.).

ctxIdx	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**
Component	luma																										
TB size	All	4×4								8×8												16×16 & 32×32					
Scan	All	All								Diagonal						Vertical/Horizontal						Diagonal					
Subblock position	DC	Only one subblock								DC subblock			Non-DC subblock			DC subblock			Non-DC subblock			DC subblock			Non-DC subblock		
SubIdx	DC	1	2	3	4	5	6	7	8	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2

ctxIdx	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**
Component	chroma														
TB size	All	4×4								8×8			16×16		
Scan	All	All scans								Diagonal					
Subblock position	DC	Only one subblock								All subblocks			All subblocks		
SubIdx	DC	1	2	3	4	5	6	7	8	0	1	2	0	1	2

for each subblock, as will be explained in more detail below. In the third scan pass, the sign of each significant level is signaled with the possible exception of the last non-zero coefficient in the subblock in reverse scanning order, as will be discussed in more detail in Sect. 6.5.2. Finally, in the last and fifth scan pass, the remaining information of absolute levels in the subblock (if present) is transmitted by using the syntax element `coeff_abs_level_remaining` (ALRem), as will be further detailed in Sect. 6.5.1 below.

6.5.1 Coding of absolute level

Coding of absolute levels requires the choice of suitable binarization schemes and, for selected bin indices, the choice of suitable context models. According to the design considerations, as discussed in Sect. 3, both aspects of coding efficiency and throughput have been properly addressed by the revised CABAC design of HEVC. This is especially true for the coding of absolute levels which typically contribute the dominant portion to the total number of generated bins. In the following, we will first elaborate on how the specific binarization scheme for absolute levels in HEVC has been designed. Then, in the second part of this subsection, we will present the context selection rules applied to the (few remaining) regular coded bins of absolute levels, unless not already done so in Sect. 6.4.1 and 6.4.3.

Conceptually, the binarization of an absolute level, denoted as z in the following, relies on a concatenated application of three binarization processes [62, 58, 24]: truncated unary (TrU), k-th order truncated Rice (TRk), and $(k+1)$-th order Exp-Golomb (EGk). Fig. 22 illustrates this binarization scheme for arbitrary z along the (discrete) number line. There are two thresholding parameters B_0, B_1 with $B_0 < B_1$

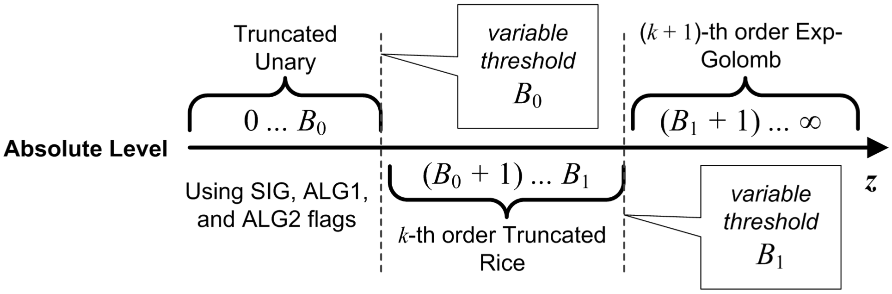


Fig. 22: Illustration of the adaptive binarization scheme for absolute levels in HEVC consisting of a concatenation of the three elementary binarizations TrU, TRk, and EGk, the latter two with varying order k and $k+1$, respectively ($0 \leq k \leq 4$). The two variable thresholds B_0 and B_1 specify the (variable) transition points between them.

which separate the 3 regions from one another for application of each of the three binarization processes and which also determine the truncation parameters cMax(TrU) $= B_0 + 1$ and cMax(TRk) $= B_1 - B_0$. The selection of the two parameters B_0, B_1 together with the choice of the parameter k is performed in a backward-adaptive manner for each subblock in such a way that the resulting bin strings are already close to a minimum-redundancy prefix code for the collection of all absolute levels z in each subblock. As a consequence, the majority of resulting bins can be simply bypass coded without compromising coding efficiency.

For each subblock, the initialization and adaptation processes for the parameters B_0, B_1, and k is performed, as follows. Before starting the processing of an subblock, the parameter k is set equal to 0, whereas B_0 is set equal to 2. The second thresholding parameter B_1 depends on k and B_0 by the fixed relation $B_1 = 4 \cdot 2^k + B_0$, which means that B_1 is adapted whenever B_0 or k are changed. For each scan position in the subblock processing, the absolute level z is evaluated after encoding/decoding and B_0 is decremented by 1 after the first occurrence of $z > 1$, which corresponds to the first scan position in the subblock for which an ALG2 flag is signaled. A further adaptation of B_0 to its minimum value of 0 is performed after $z > 0$, i.e., after an ALG1 flag occurs eight times in the subblock. The parameter k is set to $\min(k+1, 4)$ after each scan position for which the corresponding absolute level z fulfills the condition $z > 3 \cdot 2^k$. Note that according to this adaptation rule, k can take integer values from 0 to 4 inclusive. Tab. 12 and 13 show example binarizations for two different configurations of the parameters B_0, B_1, and k. Please note that the result of the binarization for z can also be interpreted as a concatenation of a unary prefix and, if present, a fixed-length suffix for different ranges of z [24]. Tab. 14 shows the corresponding binarization of ALRem, which has a maximum bin length of 32 [18].

As already indicated above, the signaling of the absolute level z involves four different syntax elements, given as `sig_coeff_flag` (SIG), `coeff_abs_level_greater1_flag` (ALG1), `coeff_abs_level_greater2_flag` (ALG2), and

Table 12: Binarization of the absolute level z for the choice of parameters $B_0 = 2$, $B_1 = 6$, and $k = 0$, corresponding to a concatenation of TrU with cMax=3, zero-order Truncated Rice (TRk) with cMax=4, and first-order Exp-Golomb (EGk).

	TrU			**TRk**				**EGk**				
	cMax=3			$k = 0$; cMax=4				$k + 1 = 1$				
z	SIG	ALG1	ALG2	0	1	2	3	0	1	2	3	…
0	0											
1	1	0										
$B_0 =$ 2	1	1	0									
3	1	1	1	0								
4	1	1	1	1	0							
5	1	1	1	1	1	0						
$B_1 =$ 6	1	1	1	1	1	1	0					
7	1	1	1	1	1	1	1	0	0			
8	1	1	1	1	1	1	1	0	1			
9	1	1	1	1	1	1	1	1	0	0	0	
10	1	1	1	1	1	1	1	1	0	0	1	
11	1	1	1	1	1	1	1	1	0	1	0	
…	…	…	…	…	…	…	…	…	…	…	…	…

Table 13: Binarization of the absolute level z for the choice of parameters $B_0 = 1$, $B_1 = 9$, and $k = 1$, corresponding to a concatenation of TrU with cMax=2, first-order Truncated Rice (TRk) with cMax=8, and second-order Exp-Golomb (EGk).

	TrU		**TRk**					**EGk**			
	cMax=2		$k = 1$; cMax=8					$k + 1 = 2$			
z	SIG	ALG1	0	1	2	3	4	0	1	2	…
0	0										
$B_0 =$ 1	1	0									
2	1	1	0	0							
3	1	1	0	1							
4	1	1	1	0	0						
5	1	1	1	0	1						
6	1	1	1	1	0	0					
7	1	1	1	1	0	1					
8	1	1	1	1	1	0	0				
$B_1 =$ 9	1	1	1	1	1	0	1				
10	1	1	1	1	1	1		0	0	0	
11	1	1	1	1	1	1		0	0	1	
…	…	…	…	…	…	…		…	…	…	…

Table 14: An alternative representation of `coeff_abs_level_remaining` (ALRem) binarization as a concatenation of a unary prefix and fixed length suffix. The suffix has a length of k bins when the value N of ALRem is less than $(3 << k)$; otherwise, it has a length of $\lfloor \log_2(((N-(3<<k)) >> k)+1) \rfloor + k$ bins. In the table below, the suffix bins are shown in terms of x and **C** , where each x represents a bin, and **C** represents a fixed length bin string of length k.

N	Prefix bins	Suffix bins	Prefix length	Suffix length	Total length	Max k
$0 \sim 2 \cdot 2^k - 1$	0	**C**	1	k	1+k	4
$1 \cdot 2^k \sim 2 \cdot 2^k - 1$	10	**C**	2	k	2+k	4
$2 \cdot 2^k \sim 3 \cdot 2^k - 1$	110	**C**	3	k	3+k	4
$2^k \cdot (2^0+2) \sim 2^k \cdot (2^1+2) - 1$	1110	**C**	4	k	4+k	4
$2^k \cdot (2^1+2) \sim 2^k \cdot (2^2+2) - 1$	11110	x**C**	5	1+k	6+k	4
$2^k \cdot (2^2+2) \sim 2^k \cdot (2^3+2) - 1$	111110	xx**C**	6	2+k	8+k	4
$2^k \cdot (2^3+2) \sim 2^k \cdot (2^4+2) - 1$	1111110	xxx**C**	7	3+k	10+k	4
$2^k \cdot (2^4+2) \sim 2^k \cdot (2^5+2) - 1$	11111110	xxxx**C**	8	4+k	12+k	4
$2^k \cdot (2^5+2) \sim 2^k \cdot (2^6+2) - 1$	111111110	xxxxx**C**	9	5+k	14+k	4
$2^k \cdot (2^6+2) \sim 2^k \cdot (2^7+2) - 1$	1111111110	xxxxxx**C**	10	6+k	16+k	4
$2^k \cdot (2^7+2) \sim 2^k \cdot (2^8+2) - 1$	11111111110	xxxxxxx**C**	11	7+k	18+k	4
$2^k \cdot (2^8+2) \sim 2^k \cdot (2^9+2) - 1$	111111111110	xxxxxxxx**C**	12	8+k	20+k	4
$2^k \cdot (2^9+2) \sim 2^k \cdot (2^{10}+2) - 1$	1111111111110	xxxxxxxxx**C**	13	9+k	22+k	4
$2^k \cdot (2^{10}+2) \sim 2^k \cdot (2^{11}+2) - 1$	11111111111110	xxxxxxxxxx**C**	14	10+k	24+k	4
$2^k \cdot (2^{11}+2) \sim 2^k \cdot (2^{12}+2) - 1$	111111111111110	xxxxxxxxxxx**C**	15	11+k	26+k	3
$2^k \cdot (2^{12}+2) \sim 2^k \cdot (2^{13}+2) - 1$	1111111111111110	xxxxxxxxxxxx**C**	16	12+k	28+k	2
$2^k \cdot (2^{13}+2) \sim 2^k \cdot (2^{14}+2) - 1$	11111111111111110	xxxxxxxxxxxxx**C**	17	13+k	30+k	1
$2^k \cdot (2^{14}+2) \sim 2^k \cdot (2^{15}+2) - 1$	111111111111111110	xxxxxxxxxxxxxx**C**	18	14+k	32+k	0

`coeff_abs_level_remaining` (ALRem), such that

$$z = \text{SIG} + \text{ALG1} + \text{ALG2} + \text{ALRem},$$

provided that the values of the corresponding syntax elements are inferred to be equal to 0, when not explicitly signaled. Note that the flags SIG, ALG1, and ALG2 represent the first and the optional second and third bin indices of the TrU part of z, respectively. ALRem corresponds to the concatenation of the TRk and EGk part of z with all of its bin values being bypass coded and with a maximum bin string length of 32 [18]. Only the values of the three flags are regular coded. However, due to the adaptation rules for B_0, ALG2 can occur only once in each subblock, while the occurrence of ALG1 is restricted to 8 scan positions per subblock at the maximum [21]. Together with the maximum of 16 SIG flags per subblock, only up to 25 regular coded bins can occur in each subblock (without accounting for CSBF). Thus, the maximum number of regular coded bins per 4×4 transform (sub-) block is reduced by a factor of about 9.6 relative to the corresponding maximum number of $16 \cdot 14 + 15 = 239$ regular coded bins for H.264/AVC CABAC (including SIG bins but without accounting for LAST) [51]. This change provides obviously the most substantial reduction to the (worst case) number of regular coded bins in the entire revision of CABAC.

The rationale behind processing SIG, ALG1, ALG2, and ALRem with individual syntax elements rather than as conventional bin indices of the adaptive binarization of z is given by the fact that all values of one syntax element in each subblock are grouped together and signaled in separate scan passes. This grouping provides essentially three advantages. First, bins in the coefficient level binarization that use the same context selection logic are grouped together to reduce the amount of speculative context selection computations, as shown in Fig. 23. Second, by grouping bypass coded bins together, the throughput advantages of bypass bins are maximized [88]. Third, the storage for (partially reconstruted) coefficient data during the parsing process at the decoder can be reduced, as further explained in Sect. 6.5.2 below. Note that the reordering of bins has no impact on coding efficiency.

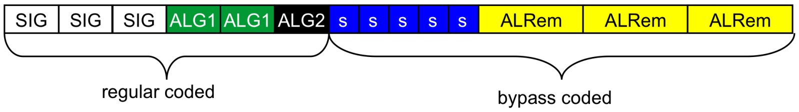


Fig. 23: Grouping same regular coded bins and bypass bins to increase throughput. s=`coeff_sign_flag`

Context modeling for coding of the regular coded bins of the absolute level is restricted to the three flags SIG, ALG1, and ALG2. Since context model selection for the SIG flag has already been introduced in Sect. 6.4.1 and 6.4.3, we will focus in the following on the two flags ALG1 and ALG2. For each of both flags, 6 sets of context models are provided: 4 sets for subblocks of the luma component and 2 sets for subblocks of the chroma component. Since only up to one ALG2 flag per subblock is encoded/decoded, each of the six ALG2 related sets contains only one context model. For the ALG1 flag, each set consists of four context models and the context increment ctxInc(ALG1) for selecting one of this four models within each set is quite similar to what is specified for the coding of the first bin of the syntax element `coeff_abs_level_minus1` in H.264/AVC (see [51] for a motivation of this design choice):

$$\text{ctxInc(ALG1)} = \begin{cases} 0, & \text{if NumG1} > 0 \\ 1 + \min(2, \text{NumT1}), & \text{otherwise} \end{cases},$$

where NumT1 denotes the accumulated number of encoded/decoded trailing 1's, i.e., absolute levels equal to 1, and NumG1 denotes the accumulated number of encoded/decoded levels with absolute value greater than 1, both computed along the reverse scanning pattern of the subblock up to (but not including) the current scan position. Note that both NumT1 and NumG1 are initialized with the value of 0 at the beginning of the subblock scan of ALG1 flags. After each encoded/decoded ALG1 flag with the value of 0, NumT1 is incremented by 1, while after each encoded/decoded ALG1 flag with the value of 1, NumG1 is incremented by 1. Fig. 24 shows the flow chart for context increment computation of ALG1.

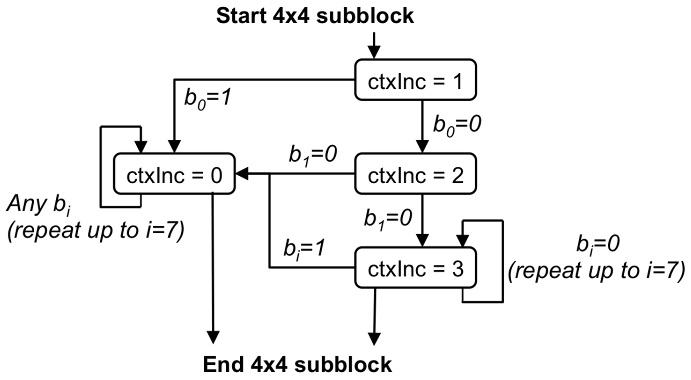


Fig. 24: Flow chart for derivation of context increment (ctxInc) for up to 8 different events b_i $(0 \le i \le 7)$ of ALG1 in a 4×4 subblock.

Since the statistics of trailing 1's may differ from subblock to subblock as well as for subblocks belonging to different components or different locations within the TB, different sets of context models are provided, both for ALG1 and ALG2, as already mentioned above. For subblocks belonging to the luma component, 2 separate sets are used for subblocks containing the DC of the TB, i.e., for the top left subblocks in a TB. Another two sets are given for luma subblocks containing no DC as well as two additional sets for chroma subblocks. Depending on the value of ctxInc(ALG1) for the last decoded ALG1 flag in the preceding subblock, the two members of each of the relevant sets related to luma DC, luma non-DC, and chroma are selected: One for the case of ctxInc(ALG1) $= 0$ and the other for the case of ctxInc(ALG1) > 0. Thus, a total number of 30 context models are used for coding of ALG1 and ALG2: $6 \cdot 4 = 24$ for ALG1 and 6 for ALG2. Interestingly enough, there was a 4× reduction (from 120 to 30) in the total number of contexts used for coding of the ALG1 and ALG2 flags during the development from HM3.0 to HM6.0 at virtually no loss in coding efficiency.

6.5.2 Coding of sign

To reduce storage cost of the coefficients, as already noted above, the transform coefficient data is grouped for every 4×4 subblock and the sign bins are bypass coded and signaled before `coeff_abs_level_remaining` bins. Before `coeff_abs_level_remaining` is added, the partial value of the coefficient level can be represented with 4 bits. Thus, CABAC in HEVC only requires storage of 4×4×4 bits for each subblock (as compared to 8×8×9 bits for a 4×4 transform block in

H.264/AVC), and the reconstructed transform coefficient level can be immediately written out once `coeff_abs_level_remaining` is parsed.

To improve coding efficiency, the optional *sign bit hiding* (SBH) technique can be used [28]. SBH is a technique to hide one bit such as, e.g., a sign of a non-zero coefficient in a group of non-zero coefficients. For this, the encoder quantizes the coefficients in the group such that the sum of their absolute level values is even or odd for the sign bit to be hidden having value 0 or 1, respectively. This inherently lossy coding technique is based on the idea that in a group of quantized coefficients, it is likely that there is at least one coefficient level for which the value can be increased or decreased by 1 with only marginally increased rate-distortion cost. This is, e.g., the case, when the unquantized coefficient was close to a quantization decision threshold, such that quantizing the coefficient to the next lower or next higher possible quantized value are both similarly good decisions.

SBH is enabled by `sign_data_hiding_enabled_flag` in the PPS and if it is enabled, it applies to each 4×4 subblock for which the number of non-zero coefficients exceeds a certain threshold. This threshold was chosen in HEVC to be a value of 3 and the sign bit to be hidden is that of the last significant scan position in the reverse scanning pattern of each subblock. The condition for SBH can be checked while parsing the significance map and thus, SBH does not have a significant impact on the entropy decoding throughput. Average bitrate savings between 0.6 to 0.9% were reported for SBH at common test conditions [104].

6.5.3 Summary of absolute level and sign coding in HEVC

Fig. 25 summarizes the last four out of up to five scan passes required for parsing the absolute levels and signs for every non-zero 4×4 subblock in HEVC.

6.6 Comparison of HEVC and H.264/AVC

Tab. 15 summarizes the differences in transform coefficient coding between HEVC and H.264/AVC as well as across different transform block sizes. In terms of throughput and memory related aspects, HEVC requires 3× fewer contexts (121 vs. 359) than H.264/AVC for transform coefficient coding. Note, however, that in H.264/AVC CABAC two separate sets of context models are used for frame-based and field-based coding of SIG and LAST. Furthermore, HEVC has a 9× lower maximum number of regular coded bins per coefficient (1.9 vs. 17.1) than H.264/AVC.

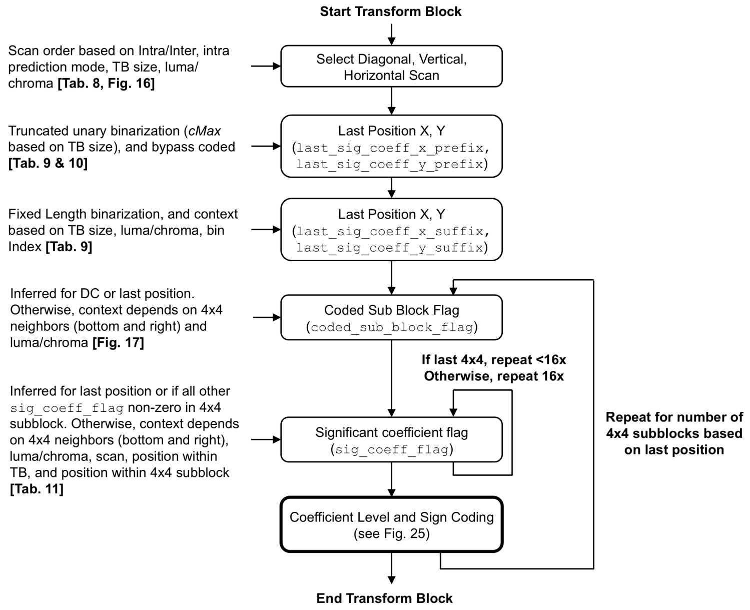

Start Transform Block
Scan order based on Intra/Inter, intra prediction mode, TB size, luma/chroma [Tab. 8, Fig. 16]
Select Diagonal, Vertical, Horizontal Scan
Truncated unary binarization (cMax based on TB size), and bypass coded [Tab. 9 & 10]
Last Position X, Y (last_sig_coeff_x_prefix, last_sig_coeff_y_prefix)
Fixed Length binarization, and context based on TB size, luma/chroma, bin Index [Tab. 9]
Last Position X, Y (last_sig_coeff_x_suffix, last_sig_coeff_y_suffix)
Inferred for DC or last position. Otherwise, context depends on 4x4 neighbors (bottom and right) and luma/chroma [Fig. 17]
Coded Sub Block Flag (coded_sub_block_flag)
If last 4x4, repeat <16x Otherwise, repeat 16x
Inferred for last position or if all other sig_coeff_flag non-zero in 4x4 subblock. Otherwise, context depends on 4x4 neighbors (bottom and right), luma/chroma, scan, position within TB, and position within 4x4 subblock [Tab. 11]
Significant coefficient flag (sig_coeff_flag)
Repeat for number of 4x4 subblocks based on last position
Coefficient Level and Sign Coding (see Fig. 25)
End Transform Block

Fig. 25: Flow chart for coding the syntax elements of absolute level minus 1 and sign for a 4x4 subblock in HEVC.

Properties		**HEVC**				**H.264/AVC**	
TB Size		**4×4**	**8×8**	**16×16**	**32×32**	**4×4**	**8×8**
Context Selection for `sig_coeff_flag` (HEVC) or `significant_coeff_flag`		Position Based	Neighbor, Scan and Position Based	Neighbor and Position Based		Position Based	
Coefficient Scanning		Intra: Diagonal, Vertical, Horizontal Inter: Diagonal		Diagonal		Zig-zag	
Number of contexts	`coded_sub_block_flag`	2 (Y), 2 (CbCr)				n/a	n/a
	`sig_coeff_flag` (HEVC) or `significant_coeff_flag` (H.264/AVC)	8 (Y), 8 (CbCr)	12 (Y), 3 (CbCr)	6 (Y), 3 (CbCr)		41×2 (Y), 18×2 (CbCr)	15×2 (Y)
		1 (Y), 1 (CbCr) DC context shared across TU sizes					
	`last_sig_coeff_x`,`y` (HEVC) or `last_significant_coeff_flag` (H.264/AVC)	15×2 (Y), 3×2 (CbCr)				41×2 (Y) 18×2 (CbCr)	9×2 (Y)
	`coeff_abs_greater1_flag`, `coeff_abs_greater2_flag` (HEVC) or `coeff_abs_level_minus1` (H.264/AVC)	4×4 + 4×1 = 20 (Y), 2×4 + 2×1 = 10 (CbCr)				30 (Y), 19 (CbCr)	10 (Y)
Maximum regular coded bins per TB	`coded_sub_block_flag`	0	2	14	62	n/a	n/a
	`significant_coeff_flag`	15	63	255	1023	15	63
	`last_sig_coeff_x, y` (HEVC) or `last_significant_coeff_flag` (H.264/AVC)	3+3 = 6	5+5 = 10	7+7 = 14	9+9 = 18	15	63
	`coeff_abs_greater1_flag`, `coeff_abs_greater2_flag` (HEVC) or `coeff_abs_level_minus1` (H.264/AVC)	(8+1) ×1 = 9	(8+1) ×4 = 36	(8+1) ×16 = 144	(8+1) ×64 = 576	4×4 ×14 = 224	8×8 ×14 = 896
Maximum regular coded bins per coefficient		1.9	1.7	1.7	1.6	17.1	16.0

Table 15: Differences between CABAC for different TB sizes in HEVC and H.264/AVC. Number of contexts for H.264/AVC includes separate models for both SIG and LAST in frame and field coding mode (denoted by "×2" in the two rightmost columns for the corresponding syntax elements).

7 Context Initialization

In HEVC, slices consist of an integer number of CTUs, which collectively form an independently decodable unit. This implies in particular that at the beginning of each slice, the parameters of all probability models must be reset to some predefined values. Typically, without any prior knowledge of the statistical nature of the source, each probability model would be initialized with the state corresponding to the uniform distribution ($p = 0.5$). However, in order to bridge the learning phase of the adaptive probability models and to enable a kind of preadaptation at different coding conditions, it was found to be beneficial to provide some more appropriate initialization value than equi-probable state for each probability model at the beginning of each slice.

Similar to H.264/AVC, CABAC in HEVC involves a quantization-parameter dependent initialization process that is invoked at the beginning of each slice. It generates an initial probability state value representing the LPS probability p_{LPS} as well as the value of the MPS v_{MPS} depending on the given initial value of the luma quantization parameter $\text{SliceQP}_{\text{Y}}$ for the slice. For that purpose, a pair of so-called *initialization parameters* is stored for each model, from which a linear relationship between $\text{SliceQP}_{\text{Y}}$ and the model probability p is derived. In contrast to H.264/AVC, the initialization parameters in HEVC do not directly represent the slope m and the offset n of the corresponding linear model. Instead, these two parameters are packed into a single 8 bit table entry in a memory-efficient way, as will be explained in more detail in the subsequent section.

For each of the three slice types I, P, and B, separate table entries are provided. However, for P and B slices the encoder can choose between the corresponding two table entries of initialization parameters and signal its choice to the decoder by use of the syntax element `cabac_init_flag`. Note that this mechanism is similar to that already available in H.264/AVC where, however, the choice between three instead of two pairs of initialization parameters is given for P and B slices [3, 51].

7.1 8-bit Design

To reduce the memory requirements for context initialization tables, it was proposed in [48] to use 8-bit values to derive the initialization parameters rather than storing the pair of 16-bit values (m,n) of the linear model directly, as in H.264/AVC. From the high nibble of the 8-bit table entry InitValue, a variable slopeIdx is derived, while the low nibble of InitValue represents the variable offsetIdx, from which the slope m and offset n of the linear model are derived using [32]

$$\begin{aligned} m &= \text{slopeIdx} \cdot 5 - 45 \\ n &= (\text{offsetIdx} << 3) - 16. \end{aligned}$$

Given the values of m and n, exactly the same initialization procedure as in H.264/AVC is performed for derivation of the parameters of each probability model [3, 51]. Note that the 8-bit design allows to cut in half the amount of storage needed for context initialization tables. Further restriction to two instead of three table entries for P and B slice types reduces the memory requirements for those tables in HEVC by at least another 12.5–15% relative to those of an 8-bit equivalent of H.264/AVC. Since there are 134 contexts for I slices and 154 for each of both slice types P and B, a total amount of 442 bytes of memory is needed for storage of all context initialization tables in HEVC.

7.2 Context Training

The main purpose of the context initialization tables is to bridge the learning phase starting from a uniform distribution, i.e., the case of no prior knowledge of the statistics of the given bin distributions, towards the well-adapted phase of the probability estimator. Assuming that after processing of a number of N_τ bins, the probability estimator that starts from $p = 0.5$ reaches such a well-adapted state, the bins for each probability model were tracked for N_τ bins for each test sequence of a training set at a particular QP and for a particular slice type. As a result, a model probability $p_{\tau,\mathrm{QP}}$ was estimated from the relative frequency obtained after coding the first N_τ bins for each probability model. This training procedure was performed separately for each QP and each of the three slice types. To finally determine the pair of parameters (m,n) that describe the assumed linear relationship between QP and model probability $p_{\tau,\mathrm{QP}}$, a simple linear regression was applied for each slice type. Note that a choice of $N_\tau = 50$ was assumed to be appropriate.

7.3 Context Memory for Wavefront Parallel Processing and Dependent Slices

For improving the parallelization and low-delay capabilities beyond the use of regular slices, as known from H.264/AVC, a partitioning of pictures into tiles, wavefronts and dependent slices have been introduced in HEVC. Since the use of regular slices implies in particular that the corresponding CABAC bitstream must be independently parsable, re-initialization of the CABAC probability models is required at the beginning of each regular slice. Although the initialization procedure, as described above, mitigates the effect of such a rigourous partitioning, the loss in coding efficiency is still too large to be acceptable for certain applications.

Wavefront parallel processing (WPP) is such a technique for picture partitioning with the focus on improving the capabilities for parallel processing at virtually no loss in coding efficiency [37, 31]. According to the WPP scheme, a picture is partitioned into rows of CTUs with each row being represented by its own CABAC bitstream which, however, is not fully independently parsable except for the bitstream belonging to first row of CTUs in a picture. Nevertheless, independent parsing and decoding of the WPP bitstreams is possible, if the processing from one CTU row to the next complies with an offset of two consecutive CTUs. This offset guarantees, on the one hand, that all spatial dependencies for the decoding process are preserved and, on the other hand, it permits inheritance of the adapted probability models from the first two CTUs in the preceding row of CTUs. The latter functionality, however, requires to store the content of all probability models after decoding the second CTU in a row. As already discussed above, the required memory depends on the slice type: for I slices 134 bytes and for P and B slices each 154 bytes of memory are needed. Note, however, that by using a proper scheduling and synchronization at the decoder, only one instance of such an additional context memory is required in addition to the N_ω context memories required for parsing and decoding N_ω CTU rows in parallel.

The same context memory handling applies also to the concept of dependent slice segments [69]. In HEVC, slices are composed of one initial independent slice segment and zero or more dependent slice segments, all of which contains an integer number of CTUs. Compared to regular slices or independent slice segments, dependent slice segments do not break the coding dependencies within the picture area to which the corresponding CTUs belong. Although each dependent slice segment has its own CABAC bitstream, the parsing of this bitstream cannot start before the parsing of the preceding dependent or independent slice segment has been finished. In particular, the content of all adapted probability models after parsing the last CTU in the preceding slice segment needs to be stored and propagated to the current dependent slice segment. Therefore, the same amount of additional context memory is required as in the WPP case. Note, however, that WPP and dependent slices, even though most often used together, are different concepts. While WPP is targeting at parallel processing, dependent slices cannot be processed in parallel and are most useful in applications requiring ultra-low delay, since each dependent slice segment can be put into a separate transport packet. Please refer to Chap. "Block Structures and Parallelism Features in HEVC" for more details.

8 Overall Performance

This section analyzes the improvements of CABAC in HEVC relative to CABAC in H.264/AVC. In the first part of this section, the impact of all relevant CABAC changes in terms of coding efficiency is experimentally evaluated, while in the second part, an assessment of its throughput implications is performed. Finally, the reduction in memory requirements is analyzed.

Simulations were performed under common test conditions set by the JCT-VC [4, 11] as well as corresponding settings for H.264/AVC JM [1]. Note that those common conditions for the HEVC reference software HM [2] are intended to reflect the typical bitstreams in applications of HEVC. During standardization of HEVC, this configuration was also used to evaluate the coding efficiency impact of proposals.

In [11], four different test cases labeled as *Intra*, *RandomAccess*, *LowDelayB*, and *LowDelayP* are specified. The Intra test case specifies that all pictures are coded as intra pictures. In the RandomAccess test case, intra pictures are inserted in regular intervals of approximately 1.1 sec in order to enable random access. As a temporal coding structure, hierarchical B pictures with groups of eight pictures are employed. Both the LowDelayB and LowDelayP test case specify that the pictures are coded in display order, so that the resulting structural encoding-decoding delay is suitable for low-delay communication applications. The latter two coding conditions differ only in the used slice type. In the LowDelayB test case, B slices are used, whereas only P slices are used in the LowDelayP test case. Note that in those low-delay test cases only one intra picture is used at the beginning of each test sequence.

The same set of test sequences as in the standardization process of HEVC has been used [11]. The test sequences are categorized into different classes, each with a particular spatial resolution. As an exception, the class labeled as *Screen content* in the following represents a special class that contains test sequences with typical screen and graphics content, but with varying spatial resolutions.

8.1 Coding Efficiency

Evaluation of coding efficiency for CABAC has been restricted to the syntax elements of transform coefficient coding. For that purpose, an extension of the residual coding scheme, specified for CABAC in H.264/AVC [51], was implemented into the HM to also cover residual coding of 16×16 and 32×32 TBs. This straightforward extension was realized by increasing the number of successive scan positions sharing the same context model for both SIG and LAST of those TBs. For the remaining syntax elements related to transform coefficient level coding, the same rules as defined for CABAC in H.264/AVC are applied [51].

Resolution and class of test sequences	**Intra**	**RandomAccess**	**LowDelayB**	**LowDelayP**
Class A: 2560×1600	-4.08	-2.86	–	–
Class B: 1920×1080	-4.18	-3.16	-3.17	-2.89
Class C: 832×416	-3.79	-2.82	-3.31	-3.13
Class D: 416×240	-4.15	-2.61	-2.43	-2.33
Class E: 1280×720	-4.92	–	-2.94	-2.69
Class F: Screen content	-7.74	-6.44	-5.79	-5.65
Average	**-4.78**	**-3.56**	**-3.54**	**-3.35**

Table 16: BD-rate performance of CABAC transform coefficient coding in HEVC compared to the extended CABAC transform coefficient coding of H.264/AVC.

Tab. 16 shows the so-called Bjøntegaard delta bit rate (BD rate) for the luma component [10] as a measure of the gain in coding efficiency obtained for the transform coefficient level coding in HEVC relative to the aforementioned straightforward CABAC extension. Overall performance gains of 3.4–4.8% in terms of averaged BD-rate savings can be attributed to the improved transform coefficient coding techniques in HEVC. The largest improvements are achieved for the Intra test case, which is mainly due to the relatively large energy of the corresponding residual signals.

Tab. 17 summarizes the individual coding efficiency impact of various adopted tools for HEVC. Note, however, that the majority of adopted tools focused on throughput improvements with minimal coding loss, as will be discussed in the following.

Tool	**HM**	**Benefit**	**BD rate**
Neighbor based context selection for SIG [102]	1.0	coding gain	-2.8% to -1.4%
Group bypass sign [14]	1.0	throughput	0.0%
Mode dependent coefficient scanning [106]	2.0	coding gain	-1.2% to -0.1%
Reduce neighboring dependency for SIG [22]	2.0	throughput	* -0.1% to 0.0%
Reduce regular coded level bins [62, 58]	3.0	throughput	-0.1% to 0.0%
Last position coding [73]	3.0	throughput	-0.1% to 0.0%
Group bypass level [88]	4.0	throughput	0.0%
Diagonal Scan [87]	4.0	throughput	-0.1% to 0.0%
CSBF & subblock scan [57, 76]	5.0	throughput	-0.1% to 0.1%
Reduce regular coded level bins per 4×4 [21]	6.0	throughput	-0.1% to 0.1%
Sign Bit Hiding [104]	6.0	coding gain	-0.9% to -0.6%
Use CSBF of neighboring subblocks for SIG [43]	7.0	throughput	0.1% to 0.2%

Table 17: Coding efficiency impact of adopted TU coding tools. Note that positive BD-rate values indicate coding loss and negative BD-rate values indicate coding gain.

8.2 Throughput Analysis

This section describes throughput of HEVC relative to H.264/AVC. The impact of the techniques, outlined in Sect. 3.3, are discussed. Analysis was also done for the worst case throughput which is defined as the case with the maximum number of bins per 16×16 coding tree unit (CTU) or macroblock. The results for both common conditions and worst case are summarized in Tab. 18 and Tab. 19, respectively.

	Common condition configurations	**Context (%)**	**Bypass (%)**	**Term (%)**
H.264/AVC	Hierarchical B	80.5	13.6	5.9
	Hierarchical P	79.4	12.2	8.4
HEVC	Intra	67.9	32.0	0.1
	LowDelayP	78.2	20.8	1.0
	LowDelayB	78.2	20.8	1.0
	RandomAccess	73.0	26.4	0.6

Table 18: Distribution of regular coded, bypass and termination bins for CABAC in H.264/AVC (JM-16.2) and HEVC (HM8.0) under common test conditions [4, 11]

Metric	**H.264/AVC**	**HEVC**	**Reduction**
Max regular coded bins	7825	882	9×
Max bypass bins	13056	13417	1×
Max total bins	20882	14301	1.5×
Number of contexts	441	154	3×
Line buffer for 4k×2k	30720	1024	30×
Coefficient storage	8×8×9-bits	4×4×3-bits	12×
Initialization Table	1746×16-bits	442×8-bits	8×

Table 19: Reduction of worst case number of bins and memory in HEVC over H.264/AVC. Note max total bins includes termination mode bins, but does not include impact of bit limit per CTU or macroblock.

8.2.1 Reduce regular coded bins

As mentioned earlier, bypass coded bins can be processed faster than regular coded bins, since they don't have data dependencies due to context selection, and their range division can be performed by a simple shift. Tab. 18 shows that the percentage of regular coded bins under common conditions is lower for HEVC than H.264/AVC. Tab. 19 also shows that in the worst case conditions, there are 9× fewer regular coded bins in HEVC than H.264/AVC. The reduction in regular coded bins is primarily due to the improved binarizations of absolute coefficient levels and components of the motion vector difference.

Using the implementation found in [103], where up to 2 regular coded bins or 4 bypass coded bins can be processed per cycle, HEVC gives 2× higher throughput than H.264/AVC under the worst case (this includes the impact of 1.5× fewer total bins in HEVC). This can also be translated into power saving using voltage scaling as mentioned earlier.

8.2.2 Group bypass coded bins

Grouping bypass bins together into longer chains increases the number of bins processed per cycle and reduces the number of cycles required to process a single bypass bin. This is a technique used in coding of syntax elements related to motion vector difference, intra mode, last position, and coefficient levels. For instance, for the Kimono sequence, encoded using the RandomAccess configuration, grouping bypass bins increases the average bypass bin run length from 2.1 to 6.4. In HEVC, under common test conditions, up to a 30% reduction in number of cycles can be achieved compared to the case of no grouping [90].

The benefit of bypass grouping can also be seen in the example of Fig. 8 and Fig. 9. If bypass grouping was not used, it would take 5 cycles to process the 5 sign bypass bins. Assuming the architecture of [103], where 4 bypass bins are processed per cycle, only 2 cycles are required to process the 5 sign bins.

8.2.3 Group bins with same context

Grouping bins with same context together is done for motion vector difference, significance map and coefficient level. As a results, fewer speculative calculations are needed to decode multiple bins per cycle since all bins that use the same logic for context selection are grouped together.

Fig. 3 showed the speculation required when `significant_coeff_flag` and `last_significant_coeff_flag` are interleaved in H.264/AVC. In HEVC, no speculation is required for significance map as shown in Fig. 26. Thus for this example, the number of operations are reduced from 14 to 5.

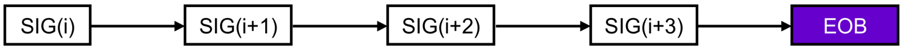


Fig. 26: No context speculation is required to achieve 5× parallelism when processing the 4×4 significance map in HEVC. i = coefficient position; EOB = end of block; SIG = `sig_coeff_flag`

8.2.4 Reduce context selection dependencies

Context selection dependencies were reduced such that coding gains could be achieved without significant penalty to throughput. For instance, the last significant coefficient position information is sent before the SIG flag to remove a tight bin to bin data dependency. Relative to HM1.0, the neighboring dependencies for SIG were reduced from 10 to 5 neighboring SIG bins, and then further modified to only depend on neighboring 4×4 subblocks. The remaining context selection for SIG is only based on its position within the block as in H.264/AVC.

8.2.5 Reduce total number of bins

When comparing the total number of bins in the worst case, and thus the throughput requirement, HEVC has 1.5× fewer bins than H.264/AVC. Assuming the same number of cycles per bin are required, HEVC can run at a 1.5× lower clock rate at a lower voltage for 50% power savings assuming linear scaling with voltage and frequency, or it can process at a bin rate that is 1.5× faster than H.264/AVC.

8.2.6 Reduce parsing dependencies

Parsing dependencies were removed or reduced such that coding gains could be achieve without significantly sacrificing throughput. Removing the parsing dependency for merge and mvp enables parsing to be mostly decoupled from the reconstruction process, as it is the case for H.264/AVC. HEVC does have parsing dependencies on intra mode reconstruction, which are not present in H.264/AVC; however, efforts were made to keep intra mode reconstruction simple to avoid affecting parsing throughput.

8.2.7 Summary of throughput improvement techniques

Tab. 20 contains a summary of the techniques for throughput improvement and related standard contributions.

Technique	**PU coding**	**TU coding**
Reduce regular coded bins	[60]	[62, 58, 21]
Group bypass bins	[67, 23]	[88]
Group bins with same context	[60]	[15, 14, 73]
Reduce context modeling dependencies		[80, 86, 22, 87]
Reduce total number of bins		[18, 57]
Reduce memory requirements	[60, 95, 83, 96]	[81, 83, 100, 82, 64, 68, 20, 25, 66, 99, 8]
Reduce parsing dependencies	[108, 107]	

Table 20: Summary of throughput improvement techniques with references to related standard contributions.

8.3 Memory Requirement Reduction

This section describes how the size and bandwidth requirements of various memories in CABAC have been reduced in HEVC in order to increase throughput as well as lower implementation cost and power consumption.

8.3.1 Context memory

The motivation for context reduction was first proposed in [81], where the number of contexts was reduced for `coeff_abs_level_greater1_flag` and `coeff_abs_level_greater2_flag` without impacting coding efficiency. Subsequent proposals [66, 99, 8] were made to reduce the number of contexts for other syntax elements (e.g. `sig_coeff_flag`). HEVC uses only 154 contexts as compared to 441 (or 292 without interlaced) used in H.264/AVC as shown in Tab. 21; thus, a 3× reduction in context memory size is achieved with HEVC.

	H.264/AVC		**HEVC**
	(w/ interlace)	(w/o interlace)	
CTU/CU contexts	25	22	16
PU contexts	26	26	14
TU contexts	390	244	124
Total	**441**	**292**	**154**

Table 21: Context memory requirements for H.264/AVC (4:2:0) and HEVC.

8.3.2 Line buffer memory

The motivation to reduce the size of the line buffer in the CABAC was first proposed in [95, 97], where the line buffer size was reduced by changing the context selection for motion vector difference. Subsequent proposals [83, 96, 68, 20, 25, 60] were made to further reduce neighboring dependencies to reduce the line buffer size. Based on these optimizations, in the worst case, the line buffer only need to store the CU depth (2-bits) of the top neighbor for context selection of `split_cu_flag` for every 8×8 block, and to indicate if the top neighbor is skipped (1-bit) for context selection of `cu_skip_flag` for ever 4×4 block. Assuming a minimum CU size of 8×8 for a 4k×2k sequence, HEVC only requires a line buffer size of 1,024 bits versus 30,720 bits in H.264/AVC, which is a 30× reduction.

8.3.3 Coefficient storage

Large TB sizes have large hardware cost implications. Compared to H.264/AVC, the 16×16 and 32×32 TBs in HEVC have 4× and 16× more coefficients than an 8×8 TB, respectively, and consequently require an increase in storage cost. Several techniques were used to reduce the coefficient storage cost. First, the sign information is sent before `coeff_abs_level_remaining` such that only 3-bits storage is required per coefficient for the partial decoded value (if stored as a 2-bit number with a range from 0 to 3, and a sign bit). Second, the coefficient information is interleaved at a 4×4 subblock level, such that the fully constructed coefficient can be achieved for every subblock and be sent out to the next module [75]. Thus, only a coefficient storage of 4×4×3-bits is required in HEVC CABAC (compared with 8×8×9-bit in H.264/AVC) in order to reconstruct the coefficient levels.

8.3.4 Context initialization tables

As already discussed in Sect. 7, the memory requirements for storing the context initialization tables in HEVC have been reduced to a large extent when compared to those of H.264/AVC. Accounting for the reduction in number of contexts, number of bits per InitValue and number of InitValue sets, HEVC has an 9× smaller context initialization table than H.264/AVC.

9 Conclusions

Entropy coding was a highly active area of development throughout the HEVC standardization process with proposals for both coding efficiency and throughput improvement. The trade-off between the two requirements were carefully evaluated in multiple Core Experiments and Ad Hoc Groups [17, 13, 16, 98, 39]. Beside coding-efficiency improving technology, many techniques were incorporated to improve throughput including reducing regular coded bins, grouping bypass bins together, grouping bins that use the same contexts together, reducing context selection dependencies, and reducing the total number of signaled bins. CABAC memory requirements were also significantly reduced. The final design of CABAC in HEVC shows that by accounting for implementation cost *and* coding efficiency when designing entropy coding algorithms results in a design that can maximize processing speed and minimize area cost, while delivering high coding efficiency in the latest video coding standard.

References

1. H.264/AVC Reference Software, JM 16.2. http://iphome.hhi.de/suehring/tml/. URL http://iphome.hhi.de/suehring/tml/
2. HEVC Test Model, HM 8.0. https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-8.0/. URL http://iphome.hhi.de/suehring/tml/
3. Recommendation ITU-T H.264: Advanced Video Coding for Generic Audiovisual Services. Tech. rep., ITU-T (2003)
4. VCEG-AM91: Joint Call for Proposals on Video Compression Technology. ITU-T Q6/16 Visual Coding and ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Audio (2010)
5. Recommendation ITU-T H.265: High Efficiency Video Coding. Tech. rep., ITU-T (2013)
6. Alshina, E., Alshin, A.: JCTVC-F254: Multi-parameter probability up-date for CABAC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
7. Amonou, I., Cammas, N., Clare, G., Jung, J., Noblet, L., Pateux, S., Matsuo, S., Takamura, S., Boon, C.S., Bossen, F., Fujibayashi, S., Kanumuri, S., Suzuki, Y., Takiue, J., Tan, T.K., Drugeon, V., Lim, C.S., Narroschke, M., Nishi, T., Sasai, H., Shibahara, Y., Uchibayashi, K., Wedi, T., Wittmann, S., Bordes, P., Gomila, C., Guillotel, P., Guo, L., Franois, E., Lu, X., Sole, J., Vieron, J., Xu, Q., Yin, P., Zheng, Y.: JCTVC-A114: Video coding technology proposal by France Telecom, NTT, NTT DoCoMo, Panasonic and Technicolor. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
8. Auyeung, C., Xu, J., Korodi, G., Zan, J., He, D., Piao, Y., Alshina E.and Min, J., Park, J.: JCTVC-G1015: A combined proposal from JCTVC-G366, JCTVC-G657, and JCTVC-G768 on context reduction of significance map coding with CABAC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
9. Belyaev, E., Gilmutdinov, M., Turlikov, A.: Binary arithmetic coding system with adaptive probability estimation by “virtual sliding window”. In: ISCE ’06. 2006 IEEE Tenth International Symposium on Consumer Electronics, pp. 1–5 (2006)
10. Bjøntegaard, G.: VCEG-M33: Calculation of Average PSNR Differences between RD curves. Video Coding Experts Group (VCEG) (2001)
11. Bossen, F.: JCTVC-J1100: HM 8 Common Test Conditions and Software Reference Configurations. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
12. Bross, B., Jung, J.: JCTVC-F913:Description of Core Experiment CE13: Motion data parsing robustness and throughput. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
13. Budagavi, M.: JCTVC-B308: Tool Experiment 8: Parallel entropy coding. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
14. Budagavi, M.: JCTVC-C062: TE8: TI parallel context processing (PCP) proposal. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
15. Budagavi, M., Demircin, M.U.: JCTVC-B088: Parallel Context Processing techniques for high coding efficiency entropy coding in HEVC. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
16. Budagavi, M., Martin-Cocher, G., Segall, A.: JCTVC-D009:JCT-VC AHG report: Entropy coding. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
17. Budagavi, M., Segall, A.: JCTVC-B009: AHG report: Parallel entropy coding. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
18. Budagavi, M., Sze, V.: JCTVC-J0142: coeff_abs_level_remaining maximum codeword length reduction. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
19. Chandrakasan, A., Sheng, S., Brodersen, R.: Low-Power CMOS Digital Design. IEEE Journal of Solid-State Circuits **27**(4), 473–484 (1992). DOI 10.1109/4.126534
20. Chen, C., Lee, T.: JCTVC-F497: Simplified context model selection for block level syntax coding. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
21. Chen, J., Chien, W.J., Joshi, R., Sole, J., Karczewicz, M.: JCTVC-H0554: Non-CE1: throughput improvement on CABAC coefficients level coding. Joint Collaborative Team on Video Coding (JCT-VC) (2012)

22. Cheung, A., Lui, W.: JCTVC-D260: Parallel processing friendly simplified context selection of significance map. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
23. Chien, W.J., Chen, J., Coban, M., Karczewicz, M.: JCTVC-I0302: Intra mode coding for INTRA_NxN. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
24. Chien, W.J., Karczewicz, M., Sole, J., Chen, J.: JCTVC-I0487: On coefficient level remaining coding. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
25. Chien, W.J., Karczewicz, M., Wang, X.: JCTVC-F606: Memory and Parsing Friendly CABAC Context. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
26. Chono, K.: JCTVC-H0712: BoG Report on Intra mode coding cleanup and simplification. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
27. Chono, K., Aoki, H.: JCTVC-F046: Efficient binary representation of cu_qp_delta syntax for CABAC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
28. Clare, G., Henry, F., Jung, J.: JCTVC-G271: Sign Data Hiding. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
29. Finchelstein, D., Sze, V., Chandrakasan, A.: Multicore Processing and Efficient On-Chip Caching for H.264 and Future Video Decoders. IEEE Trans. on CSVT **19**(11), 1704–1713 (2009)
30. Fuldseth, A., Bjøntegaard, G., Budagavi, M., Sze, V.: JCTVC-G495: CE10: Core Transform Design for HEVC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
31. Gordon, C., Henry, F., Pateux, S.: JCTVC-F274: Wavefront Parallel Processing for HEVC Encoding and Decoding. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
32. Guo, L., Sole, J., Joshi, R., Karczewicz M.and Yeo, C., Tan, Y., Li, Z.: JCTVC-H0535: CE1 B3: 8-bit Linear Initialization for CABAC. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
33. Guo, X., Huang, Y.W., Lei, S.: VCEG-AK25: Ordered Entropy Slices for Parallel CABAC. Video Coding Experts Group (VCEG) (2009)
34. He, D., Korodi, G., Martin-Cocher, G., Yang, E.h., Yu, X., Zan, J.: JCTVC-A120: Video coding technology proposal by Research in Motion. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
35. Helle, P., Oudin, S., Bross, B., Marpe, D., Bici, M., Ugur, K., Jung, J., Clare, G., Wiegand, T.: Block Merging for Quadtree-Based Partitioning in HEVC. IEEE Trans. on CSVT **22**(12), 1720–1731 (2012)
36. Hellman, T., Yu, Y.: JCTVC-F341: Decoder Performance Restrictions due to Merge/MVP Index Parsing. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
37. Henry, F., Pateux, S.: JCTVC-E196: Wavefront Parallel Processing. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
38. Huang, Y.W., Alshina, E.: JCTVC-I0602: BoG report on integrated text of SAO adoptions on top of JCTVC-I0030. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
39. Joshi, R., Alshina, E., Sasai, H., Kirchhoffer, H., Lainema, J.: JCTVC-F901: Description of Core Experiment 1: Entropy Coding. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
40. Jung, J., Laroche, G.: VCEG-AC06:Competition-Based Scheme for Motion Vector Selection and Coding. Video Coding Experts Group (VCEG) (2006)
41. Karczewicz, M., Chen, P., Joshi, R., Wang, X., Chien, W.J., Panchal, R.: JCTVC-A121:Video coding technology proposal by Qualcomm. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
42. Kim, W.S., Kwon, D.K.: JCTVC-G0680: ANon-CE8: Method of visual coding artifact removal for SAO. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
43. Kumakura T. Fukushima, S.: JCTVC-I0296: Non-CE3: Simplified context derivation for significance map. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
44. Lan, C., Xu, J., Sullivan, G.J., Wu, F.: JCTVC-I0408: Intra transform skipping. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
45. Marpe, D., Blättermann, G., Wiegand, T.: VCEG-L13: Adaptive Codes for H.26L. Video Coding Experts Group (VCEG) (2001)

46. Marpe, D., Bosse, S., Bross, B., Helle, P., Hinz, T., Kirchhoffer, H., Lakshman, H., Oudin, S., Schwarz, H., Siekmann, M., Sühring, K., Winken, M., Wiegand, T.: Video Compression Using Nested Quadtree Structures, Leaf Merging and Improved Techniques for Motion Representation and Entropy Coding. IEEE Trans. on CSVT **20**(12), 1676–1687 (2010)
47. Marpe, D., Heising, G., Blättermann, G., Wiegand, T.: JVT-C061: Fast Arithmetic Coding for CABAC. Joint Video Team (JVT) (2002)
48. Marpe, D., Kirchhoffer, H., Bross, B., George, V., Nguyen, T., Preiß, M., Siekmann, M., Stegemann, J., Wiegand, T.: JCTVC-F268: Unified PIPE-Based Entropy Coding for HEVC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
49. Marpe, D., Marten, G., Cycon, H.L.: A Fast Renormalization Technique for H. 264/MPEG4-AVC Arithmetic Coding. In: 51st Internationales Wissenschaftliches Kolloquium Technische Universität Ilmenau (2006)
50. Marpe, D., Marten, G., Wiegand, T.: JVT-U084: Fast CABAC renormalization for H.264/MPEG4-AVC. Joint Video Team (JVT) (2006)
51. Marpe, D., Schwarz, H., Wiegand, T.: Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. IEEE Trans. on CSVT **13**(7), 620– 636 (2003)
52. Marpe, D., Schwarz, H., Wiegand, T.: Entropy coding in video compression using probability interval partitioning. In: Picture Coding Symposium (PCS), 2010, pp. 66–69 (2010)
53. Marpe, D., Schwarz, H., Wiegand, T.: JCTVC-A032: Novel entropy coding concept. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
54. Marpe, D., Wiegand, T.: A highly efficient multiplication-free binary arithmetic coder and its application in video coding. In: IEEE Inter. Conf. on Image Processing,, pp. 263–266 (2003)
55. McCann, K., Bross, B., Sekiguchi, S., Han, W.J.: JCTVC-C402: HEVC Test Model 1 (HM 1) Encoder Description. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
56. Nguyen, N., Ji, T., He, D., Martin-Cocher, G.: JCTVC-H0554: Non-CE1: throughput improvement on CABAC coefficients level coding. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
57. Nguyen, N., Ji, T., He, D., Martin-Cocher, G., Song, L.: JCTVC-G644: Multi-level Significant Maps for Large Transform Units. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
58. Nguyen, T.: JCTVC-E253: CE11: Coding of transform coefficient levels with Golomb-Rice codes. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
59. Nguyen, T., Marpe, D., Schwarz, H., Wiegand, T.: JCTVC-D061: CE11: Evaluation of Transform Coding tools in HE configuration. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
60. Nguyen, T., Marpe, D., Schwarz, H., Wiegand, T.: JCTVC-F455: Modified binarization and coding of MVD for PIPE/CABAC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
61. Nguyen, T., Schwarz, H., Kirchhoffer, H., Marpe, D., Wiegand, T.: Improved context modeling for coding quantized transform coefficients in video compression. In: Picture Coding Symposium (PCS), 2010, pp. 378–381 (2010)
62. Nguyen, T., Winken, M., Marpe, D., Schwarz, H., Wiegand, T.: JCTVC-D336: Reduced-complexity entropy coding of transform coefficient levels using a combination of VLC and PIPE. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
63. Peng, X., Lan, C., Xu, J., Sullivan, G.J.: JCTVC-J0237: Inter transform skipping. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
64. Piao, Y., Min, J., Alshina, E., Park, J.T.: JCTVC-G781: Reduced chroma contexts for significance map coding in CABAC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
65. Ryabko, B.Y.: Imaginary sliding window as a tool for data compression. Probl. Inform. Transm. pp. 156–163 (1996)
66. Sasai, H., Nishi, T.: JCTVC-E227: CE11: Context size reduction for the significance map. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
67. Sasai, H., Nishi, T.: JCTVC-F423: Modified MVD coding for CABAC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)

68. Sasai, H., Nishi, T.: JCTVC-F429: Modified Context Derivation for neighboring dependency reduction. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
69. Schierl, T., Goerge, V., Henkel, A., Marpe, D.: JCTVC-I0229: Dependent Slices. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
70. Seregin, V., Kim, I.K.: JCTVC-F375: Binarisation modification for last position coding. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
71. Seregin, V., Sole, J., Karczewicz, M., Wang, X., Sze, V., Budagavi, M.: JCTVC-J0098: AHG5: Bypass bins for reference index coding. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
72. Shannon, C.E.: A mathematical theory of communications **27**, 379–423 (1948)
73. Sole, J., Joshi, R., Karczewicz, M.: JCTVC-E338: CE11: Parallel Context Processing for the significance map in high coding efficiency. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
74. Sole, J., Joshi, R., Karczewicz, M.: JCTVC-F288: CE11: Unified scans for the significance map and coefficient level coding in high efficiency. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
75. Sole, J., Joshi, R., Karczewicz, M.: JCTVC-F552:CE11: Scanning of Residual Data in HE. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
76. Sole, J., Joshi, R., Karczewicz, M.: JCTVC-G323: Non-CE11: Diagonal sub-block scan for HE residual coding. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
77. Sole, J., Joshi, R., Karczewicz, M.: JCTVC-J0256:Removal of the 8x2/2x8 coefficient groups. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
78. Stegemann, J., Kirchhoffer, H., Marpe, D., Wiegand, T.: JCTVC-G547: Non-CE1: Counter-based probability model update with adapted arithmetic coding engine . Joint Collaborative Team on Video Coding (JCT-VC) (2011)
79. Sugio, T., Nishi, T.: JCTVC-F470: Parsing Robustness for Merge/AMVP. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
80. Sze, V.: JCTVC-D244: Context selection complexity in HEVC CABAC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
81. Sze, V.: JCTVC-F132: Reduction in contexts used for significant_coeff_flag and coefficient level. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
82. Sze, V.: JCTVC-F132: Reduction in contexts used for significant_coeff_flag and coefficient level. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
83. Sze, V.: JCTVC-F746: BoG report on context reduction for CABAC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
84. Sze, V., Allen, R.: JCTVC-G1017: BoG Report on Intra Mode Coding. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
85. Sze, V., Budagavi, M.: T05-SG16-C-0334: Parallel CABAC (2008)
86. Sze, V., Budagavi, M.: JCTVC-C227: Parallelization of HHI_TRANSFORM_CODING. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
87. Sze, V., Budagavi, M.: JCTVC-F129: CE11: Parallelization of HHI_TRANSFORM_CODING Fixed Diagonal Scan. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
88. Sze, V., Budagavi, M.: JCTVC-F130: Parallel Context Processing of Coefficient Level. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
89. Sze, V., Budagavi, M.: High Throughput CABAC Entropy Coding in HEVC. IEEE Trans. on CSVT **22**(12), 1778–1791 (2012). DOI 10.1109/TCSVT.2012.2221526
90. Sze, V., Budagavi, M.: A Comparison of CABAC Throughput for HEVC/H.265 vs. AVC/H.264. In: IEEE Workshop on Signal Processing Systems (2013)
91. Sze, V., Budagavi, M., Chandrakasan, A.: VCEG-AL21: Massively Parallel CABAC. Video Coding Experts Group (VCEG) (2009)
92. Sze, V., Budagavi, M., Chandrakasan, A., Zhou, M.: Parallel CABAC for Low Power Video Coding. In: IEEE Inter. Conf. on Image Processing,, pp. 2096–2099 (2008)
93. Sze, V., Budagavi, M., Demircin, M.U.: VCEG-AJ31: CABAC throughput requirements for real-time decoding. Video Coding Experts Group (VCEG) (2008)

94. Sze, V., Budagavi, M., Seregin, V., Sole, J., Karczewicz, M.: JCTVC-J0089: AHG5: Bin reduction for delta QP coding. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
95. Sze, V., Chandrakasan, A.P.: JCTVC-E324: Joint Algorithm-Architecture Optimization of CABAC. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
96. Sze, V., Chandrakasan, A.P.: JCTVC-F133: Simplified MVD context selection (Extension of JCTVC-E324). Joint Collaborative Team on Video Coding (JCT-VC) (2011)
97. Sze, V., Chandrakasan, A.P.: Joint algorithm-architecture optimization of CABAC to increase speed and reduce area cost. In: IEEE International Conf. on Acoustics, Speech and Signal Processing, pp. 1577–1580 (2011)
98. Sze, V., Panusopone, K., Chen, J., Nguyen, T., Coban, M.: JCTVC-C511: Description of Core Experiment 11: Coefficient Scanning and Coding. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
99. Sze, V., Sasai, H.: JCTVC-E489: Modification to JCTVC-E227 in CE11 for reduced dependency with MDCS. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
100. Terada, K., Sasai, H., Nishi, T.: JCTVC-H0290: Non-CE11:Simplification of context selection for significant_coeff_flag. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
101. Ugur, K., Saxena, A.: JCTVC-J0021: CE1: Summary Report of Core Experiment on Intra Transform Mode Dependency Simplifications. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
102. Winken, M., Bosse, S., Bross, B., Helle, P., Hinz, T., Kirchhoffer, H., Lakshman, H., Marpe, D., Oudin, S., Preiß, M., Schwarz, H., Siekmann, M., Sühring, K., Wiegand, T.: JCTVC-A116: Description of video coding technology proposal by Fraunhofer HHI. Joint Collaborative Team on Video Coding (JCT-VC) (2010)
103. Yang, Y.C., Guo, J.I.: High-Throughput H.264/AVC High-Profile CABAC Decoder for HDTV Applications. IEEE Trans. on CSVT **19**(9), 1395 –1399 (2009). DOI 10.1109/TCSVT.2009.2020340
104. Yu, X., Wang, J., He, D., Martin-Cocher, G., Campbell, S.: JCTVC-H0481: Multiple Sign Bits Hiding. Joint Collaborative Team on Video Coding (JCT-VC) (2012)
105. Zhao, J., Segall, A.: COM16-C405: Entropy slices for parallel entropy decoding (2008)
106. Zheng, Y., Coban, M., Wang, X., Sole, J., Joshi, R., Karczewicz, M.: JCTVC-D393: CE11: Mode Dependent Coefficient Scanning. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
107. Zhou, M., Sze, V.: JCTVC-E118: A study on HM2.0 bitstream parsing and error resiliency issue. Joint Collaborative Team on Video Coding (JCT-VC) (2011)
108. Zhou, M., Sze, V., Mastuba, Y.: JCTVC-F068: A study on HEVC parsing throughput issue. Joint Collaborative Team on Video Coding (JCT-VC) (2011)