High Throughput CABAC Entropy Coding in HEVC

Vivienne Sze, *Member, IEEE,* and Madhukar Budagavi, *Senior Member, IEEE*

***Abstract*—Context-adaptive binary arithmetic coding (CABAC) is a method of entropy coding first introduced in H.264/AVC and now used in the newest standard High Efficiency Video Coding (HEVC). 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 coding efficiency and throughput were considered. This paper highlights the key techniques that were used to enable HEVC to potentially achieve higher throughput while delivering coding gains relative to H.264/AVC. These techniques include reducing context coded bins, grouping bypass bins, grouping bins with the same context, reducing context selection dependencies, reducing total bins, and reducing parsing dependencies. It also describes reductions to memory requirements that benefit both throughput and implementation costs. Proposed and adopted techniques up to draft international standard (test model HM-8.0) are discussed. In addition, analysis and simulation results are provided to quantify the throughput improvements and memory reduction compared with H.264/AVC. In HEVC, the maximum number of context-coded bins is reduced by 8×, and the context memory and line buffer are reduced by 3× and 20×, respectively. This paper illustrates that accounting for implementation cost when designing video coding algorithms can result in a design that enables higher processing speed and lowers hardware costs, while still delivering high coding efficiency.**

***Index Terms*—Context-adaptive binary arithmetic coding (CABAC), entropy coding, high-efficiency video coding (HEVC), video coding.**

I. Introduction

HIGH EFFICIENCY Video Coding (HEVC) is currently being developed by the Joint Collaborative Team for Video Coding (JCT-VC). It is expected to deliver up to a 50% higher coding efficiency compared to its predecessor H.264/AVC. HEVC uses several new tools for improving coding efficiency, including larger block and transform sizes, additional loop filters, and highly adaptive entropy coding. 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 next-generation video coding to handle the demand for higher resolution and frame rates.

Manuscript received April 16, 2012; revised July 7, 2012; accepted August 21, 2012. Date of publication October 2, 2012; date of current version January 8, 2013. This paper was recommended by Associate Editor F. Wu.

The authors are with Texas Instruments, Dallas, TX 75243 USA (e-mail: sze@alum.mit.edu; madhukar@ti.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2012.2221526

Context-adaptive binary arithmetic coding (CABAC) [1] is a form of entropy coding used in H.264/AVC [2] and also in HEVC [3]. While CABAC provides high coding efficiency, its data dependencies cause it to be a throughput bottleneck for H.264/AVC video codecs [4]. This makes it difficult to support the growing throughput requirements for next-generation video codecs. Furthermore, since high throughput can be traded off for power savings using voltage scaling [5], the serial nature of CABAC limits the battery life for video codecs that reside on mobile devices. This limitation is a critical concern, as a significant portion of the video codecs today are battery operated. Accordingly, both coding efficiency and throughput improvement tools, and the tradeoff between these requirements, were investigated in the standardization of entropy coding for HEVC. The tradeoff between coding efficiency and throughput exists, since dependencies are a result of removing redundancy, which improves coding efficiency; however, dependencies make parallel processing difficult, which degrades throughput.

This paper describes how CABAC entropy coding has evolved from H.264/AVC to HEVC (Draft International Standard, HM-8.0) [3], [6]. While both coding efficiency and throughput improvement tools are discussed, the focus of this paper will be on tools that increase throughput while maintaining coding efficiency. Section II provides an overview of CABAC entropy coding. Section III explains the cause of the throughput bottleneck. Section IV describes several key techniques used to improve the throughput of the CABAC engine. Sections V and VI describe how these techniques are applied to prediction unit (PU) coding and transform unit (TU) coding, respectively. Section VII compares the overall CABAC throughput and memory requirements of H.264/AVC and HEVC for both common conditions and worst-case conditions.

II. CABAC Entropy Coding

Entropy coding is a form of lossless compression used at the last stage of video encoding (and the first stage of video decoding), after the video has been reduced to a series of syntax elements. Syntax elements describe how the video sequence can be reconstructed at the decoder. This includes the method of prediction (e.g., spatial or temporal prediction, intra prediction mode, and motion vectors) and prediction error, also referred to as residual. Table I shows the syntax elements used in HEVC and H.264/AVC. These syntax elements describe properties of the coding unit (CU),

TABLE I
CABAC CODED SYNTEX ELEMENTS IN HEVC AND H.264/AVC

		HEVC	H.264/AVC
Coding unit (CU)	Block structure and quantization	split_cu_flag, pred_mode_flag, part_mode, pcm_flag, cu_transquant_bypass_flag, skip_flag, cu_qp_delta_abs, cu_qp_delta_sign, end_of_slice_flag	mb_type, sub_mb_type, mb_skip_flag, mb_qp_delta, end_of_slice_flag, mb_field_decoding_flag
Prediction unit (PU)	Intra mode coding	prev_intra_luma_pred_flag, mpm_idx, rem_intra_luma_pred_mode, intra_chroma_pred_mode	prev_intra4x4_pred_mode_flag, prev_intra8 × 8_pred_mode_flag, rem_intra4x4_pred_mode, rem_intra8x8_pred_mode, intra_chroma_pred_mode
	Motion data	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)	Transform coefficient coding	no_residual_syntax_flag, split_transform_flag, cbf_luma, cbf_cb, cbf_cr, transform_skip_flag, last_significant_coeff_x_prefix, last_significant_coeff_y_prefix, last_significant_coeff_x_suffix, last_significant_coeff_y_suffix, coded_sub_block_flag, significant_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
Loop filter (LF)	Sample adaptive offset (SAO) parameters	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), transform unit (TU), and loop filter (LF) of a coded block of pixels. For a CU, the syntax elements describe the block structure and whether the CU is inter or intra predicted. For a PU, the syntax elements describe the intra prediction mode or a set of motion data. For a TU, the syntax elements describe the residual in terms of frequency position, sign, and magnitude of the quantized transform coefficients. The LF syntax elements are sent once per largest coding unit (LCU), and describe the type (edge or band) and offset for sample adaptive offset in-loop filtering.

Arithmetic coding is a type of entropy coding that can achieve compression close to the entropy of a sequence by effectively mapping the symbols (i.e., syntax elements) to codewords with a noninteger number of bits. In H.264/AVC, CABAC provides a 9% to 14% improvement over the Huffman-based CAVLC [1]. In an early test model for HEVC (HM-3.0), CABAC provides a 5%–9% improvement over CAVLC [7].

CABAC involves three main functions: binarization, context modeling, and arithmetic coding. Binarization maps the syntax elements to binary symbols (bins). Context modeling estimates the probability of the bins. Finally, arithmetic coding compresses the bins to bits based on the estimated probability.

A. Binarization

Several different binarization processes are used in HEVC, including unary (U), truncated unary (TU), kth-order Exp-Golomb (EGk), and fixed length (FL). These forms of 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. An example is also provided in Table II.

1) 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.
2) Truncated unary coding has one less bin than unary coding by setting a maximum on the largest possible value of the syntax element (cMax). When N+1<cMax, the signaling is the same as unary coding. However, when N+1=cMax, all bins are 1. The decoder searches for a 0 up to cMax bins to determine when the syntax element is complete.
3) kth order Exp-Golomb is a type of universal code. The distribution can be changed based on the k parameter. More details can be found in [1].
4) Fixed length uses a fixed number of bins, ceil($\log_2$(cMax+1)), with most significant bits signaled before least significant bits.

TABLE II
EXAMPLE OF DIFFERENT BINARIZATIONS USED IN HEVC

N	Unary (U)	Truncated unary (TU)	Exp-Golomb (EGk)	Fixed length (FL)
		cMax=7	k=0	cMax=7
0	0	0	1	000
1	10	10	010	001
2	110	110	011	010
3	1110	1110	00100	011
4	11110	11110	00101	100
5	111110	111110	00110	101
6	1111110	1111110	00111	110
7	11111110	1111111	0001000	111

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 elements (e.g., the binarization of coeff_abs_level_remaining depends on the previous coefficient levels) or slice pa-

rameters that indicate if certain modes are enabled (e.g., the binarization of partition mode, part_mode, depends on whether asymmetric motion partition is enabled). The majority of the syntax elements use the binarization processes listed in Table II, or some combination of them (e.g., coeff_abs_level_remaining uses U(prefix)+FL(suffix) [8]; cu_qp_delta_abs uses TU(prefix)+EG0(suffix) [9]). However, certain syntax elements (e.g., part_mode and intra_chroma_pred_mode) use custom binarization processes.

B. Context Modeling

Context modeling provides an accurate probability estimate required to achieve high coding efficiency. Accordingly, it is highly adaptive and different context models can be used for different bins and the probability of that context model is updated based on the values of the previously coded bins. Bins with similar distributions often share the same context model. The context model for each bin can be selected based on the type of syntax element, bin position in syntax element (binIdx), luma/chroma, neighboring information, etc. A context switch can occur after each bin. The probability models are stored as 7-b entries [6-b for the probability state and 1-b for the most probable symbol (MPS)] in a context memory and addressed using the context index computed by the context selection logic. HEVC uses the same probability update method as H.264/AVC; however, the context selection logic has been modified to improve throughput.

C. Arithmetic Coding

Arithmetic coding is based on recursive interval division. A range, with an initial value of 0 to 1, is divided into two subintervals based on the probability of the bin. The encoded bits provide an offset that, when converted to a binary fraction, selects one of the two subintervals, which indicates the value of the decoded bin. After every decoded bin, the range is updated to equal the selected subinterval, and the interval division process repeats itself. The range and offset have limited bit precision, so renormalization is required whenever the range falls below a certain value to prevent underflow. Renormalization can occur after each bin is decoded.

Arithmetic coding can be done using an estimated probability (context coded), or assuming equal probability of 0.5 (bypass coded). For bypass coded bins, the division of the range into subintervals can be done by a shift, whereas a look up table is required for the context coded bins. HEVC uses the same arithmetic coding as H.264/AVC.

III. Throughput Bottleneck

CABAC is a well-known throughput bottleneck in the video codec implementations (particularly, at the decoder). 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.

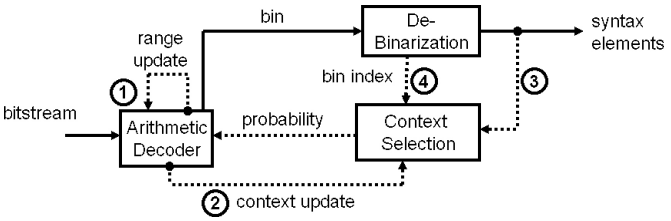

Fig. 1. Three key operations in CABAC: binarization, context selection, and arithmetic coding. Feedback loops in the decoder are highlighted with dashed lines.

Fig. 1 highlights the feedback loops in the CABAC decoder. Below is a list and description of these feedback loops.

1) The updated range is fed back for recursive interval division.
2) The updated context is fed back for an accurate probability estimate.
3) The context selection depends on the type of syntax element. At the decoder, the decoded bin 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 bin may be used to determine which syntax element to decode next.
4) The context selection also depends on the bin position in the syntax element (binIdx). At the decoder, the decoded bin 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.

Note that the context update and range update feedback loops are simpler than the context selection loops 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 [10]. The amount of speculation can grow exponentially with the number of parallel bins which limits the throughput that can be achieved [11]. Fig. 2 shows an example of the speculation tree for significance map in H.264/AVC. Thus, the throughput bottleneck is primarily due to the context selection dependencies.

IV. Techniques to Improve Throughput

Several techniques were used to improve the throughput of CABAC in HEVC. 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 paper.

1) *Reduce Context Coded Bins:* Throughput is limited for context coded bins due to the data dependencies described in Section III. However, it is easier to process bypass coded bins in parallel since they do not have the data dependencies related to context selection (i.e., feedback loops 2, 3, and 4 in Fig. 1). In addition, arithmetic coding for bypass bins is simpler as it only requires a right shift versus a table look up for context coded bins. Thus, the throughput can be improved by reducing the number of context coded bins and using bypass coded bins instead [12]–[14].

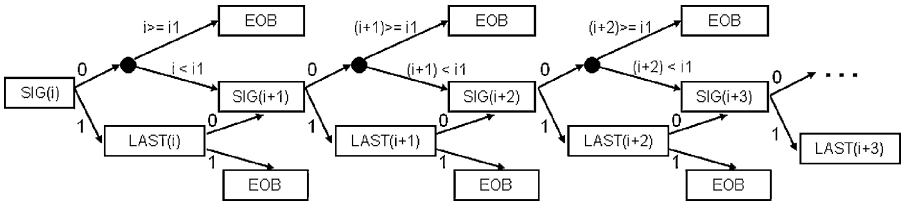

Fig. 2. Context speculation required to achieve 5× parallelism when processing the significance map in H.264/AVC. i = coefficient position, $i1$ = MaxNumCoeff(BlockType)−1, EOB = end of block, SIG = significant_coeff_flag, LAST = last_significant_coeff_flag.

2) *Group Bypass Coded Bins:* Multiple bypass bins can be processed in the same cycle 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 [15]–[17].

3) *Group Bins With the Same Context:* Processing multiple context coded bins in the same cycle often requires speculative calculations for context selection. The amount of speculative computations increases if bins using different contexts and context selection logic are interleaved, since various 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 selection logic are grouped together so that they are likely to be processed in the same cycle [18]–[20]. This also reduces context switching resulting in fewer memory accesses, which also increases throughput and reduces power consumption. This technique was first introduced in [18] and referred to as parallel context processing (PCP) throughout the standardization process.

4) *Reduce Context Selection Dependencies:* Speculative computations are required for multiple bins per cycle decoding due to the dependencies in the context selection. Reducing these dependencies simplifies the context selection logic and reduces the amount of speculative calculations required to process multiple bins in parallel [11], [21], [22].

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 binarization, inferring the value of some bins, and sending higher level flags to avoid signaling redundant bins [23], [24].

6) *Reduce Parsing Dependencies:* As parsing with CABAC is already a tight bottleneck, it is important to minimize any dependency on other video coding modules, which could cause the CABAC to stall [25]. Ideally, the parsing process should be decoupled from all other processing.

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 [26], [27].

V. Prediction Unit Coding

The PU syntax elements describe how the prediction is performed in order to reconstruct the pixels. For inter prediction, the motion data are described by merge flag (merge_flag), merge index (merge_idx), prediction direction (inter_pred_idc), reference index (ref_idx_l0, ref_idx_l1), motion vector predictor flag (mvp_l0_flag, mvp_l1_flag) and motion vector difference (abs_mvd_greater0_flag, abs_mvd_greater1_flag, abs_mvd_minus2, mvd_sign_flag). For intra prediction, the intra prediction mode is described by prediction flag (prev_intra_luma_pred_flag), most probable mode index (mpm_idx), remainder mode (rem_intra_luma_pred_mode) and intra prediction mode for chroma (intra_chroma_pred_mode).

Coding efficiency improvements have been made in HEVC for both motion data coding and intra mode coding. While H.264/AVC uses a single motion vector predictor (unless direct mode is used) or most probable mode, HEVC uses multiple candidate predictors and an index or flag is signaled to select the predictor. This section will discuss how to avoid parsing dependencies for the various methods of prediction and other throughput improvements.

A. Motion Data Coding

In HEVC, merge mode enables motion data (e.g., 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 nonzero is reached or when the number of bins is equal to the cMax, the max allowed number of bins.

1) *Removing Parsing Dependencies for Merge:* Determining how to set cMax involved evaluating the throughput and coding efficiency tradeoffs in a core experiment [28]. 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 [25], [29].

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 and does not depend on list size. To compensate for the coding loss due to the fixed cMax,

combined and zero merging candidates are added when the list size is less than cMax as described in [30]. This ensures that the list is never empty and that merge_flag is always signaled [31].

2) *Removing Parsing Dependencies for Motion Vector Prediction:* If merge mode is not used, then the motion vector is predicted from its neighboring blocks and the difference between motion vector prediction (mvp) and motion vector (mv), referred to as *motion vector difference* (mvd), is signaled as

$$\text{mvd} = \text{mv} - \text{mvp}.$$

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 [32]. 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 vector is used as the default candidate.

3) *Reducing Context Coded Bins:* In HEVC, improvements were also made on the coding process of mvd itself. In H.264/AVC, the first 9 bins of mvd are context coded truncated unary bins, followed by bypass coded third-order Exp-Golomb bins. In HEVC, the number of context coded bins for mvd is significantly reduced [13]. Only the first two bins are context coded (abs_mvd_greater0_flag, abs_mvd_greater1_flag), followed by bypass coded first-order Exp-Golomb bins (abs_mvd_minus2).

4) *Reducing Memory Requirements:* 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-b storage per neighboring motion vector, which accounts 24 576 of the 30 720-b CABAC line buffer needed to support a $4k \times 2k$ sequence. [27] highlighted the need to reduce the line buffer size in HEVC by modifying the context selection logic. It proposed that the motion vector for each neighbor be first compared to a threshold of 16, and then use the sum of the comparison for context selection, which would reduce the storage to 1-b per neighboring motion vector. This was further extended in [33] and [34], where 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).

5) *Grouping Bypass Bins:* To maximize the impact of fast bypass coding, the bypass coded bins for both the x and y components of mvd are grouped together in HEVC [16].

B. Intra Mode Coding

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

In HEVC, additional most probable modes 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, +1 or −1 angular mode) can be added if the left and top neighbors are the same or unavailable. prev_intra_pred_mode_flag is signaled to indicate whether one of the most probable modes is used. If a most probable mode 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 [35], [36].

1) *Reducing Context Coded Bins:* The number of context coded bins was reduced for intra mode coding in HEVC. In H.264/AVC, the remainder mode is a 7-bin value where the first bin is context coded, while in HEVC, the remainder mode is a 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.

2) *Grouping Bypass Bins:* To maximize the impact of fast bypass coding, the bypass coded bins for intra mode within a CU are grouped together in HEVC [17].

TABLE III
DIFFERENCES BETWEEN PU 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 context coded bins per PU	2	12	2	7	98

C. Summary of Differences Between HEVC and H.264/AVC

The differences between H.264/AVC and HEVC are summarized in Table III. 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 $8\times$ fewer maximum context coded bins per PU than H.264/AVC. HEVC also requires $1.8\times$ fewer contexts for PU syntax elements than H.264/AVC.

VI. Transform Unit Coding

In video coding, both intra and inter prediction are used to reduce the amount of data that needs to be transmitted. Rather than sending the pixels, the prediction error is transmitted. This prediction error is transformed from spatial to frequency domain to leverage energy compaction properties, and after quantization, it can be represented in terms of a few coefficients. The method of signaling the value and the frequency position of these coefficients is referred to as transform coefficient coding. For regions with many edges (e.g., screen content coding), coding gains can be achieved by skipping the transform from spatial to frequency domain [37], [38]; when transform is skipped, the prediction error is coded in the same manner as transform coefficient coding (i.e., the spatial error is coded as transform coefficients).

Syntax elements of the transform unit account for a significant portion of the bin workload as shown in Table IV. At the same time, the transform coefficients also account for a significant portion of the total bits of a compressed video, and as a result the compression of 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 [39].

It is also important to note that HEVC supports more transform sizes than H.264/AVC; H.264/AVC uses 4×4 and 8×8 transforms, where as HEVC uses 4×4, 8×8, 16×16, and 32×32 transforms. While these larger transforms provide significant coding gains, they also have implications in terms of memory storage as this represents an increase of $4\times$ to $16\times$ in the number of coefficients that need to be stored per transform unit (TU).

In CABAC, the position of the coefficients is transmitted in the form of a *significance map*. Specifically, the significance map indicates the location of the nonzero coefficients. The *coefficient level* information is then only transmitted for the coefficients with values greater than one, while the coefficient sign is transmitted for all nonzero coefficients.

This section describes how transform coefficient coding evolved from H.264/AVC to the first test model of HEVC (HM-1.0) to the Draft International Standard of HEVC (HM-8.0), 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, Figs. 3 and 4 show examples of transform coefficient coding in H.264/AVC and HEVC (HM-8.0), respectively.

A. Significance Map

In H.264/AVC, the significance map is signaled by transmitting a significant_coeff_flag (SCF) for each position to indicate whether the coefficient is nonzero. The positions are processed in an order based on a zig–zag scan. After each nonzero

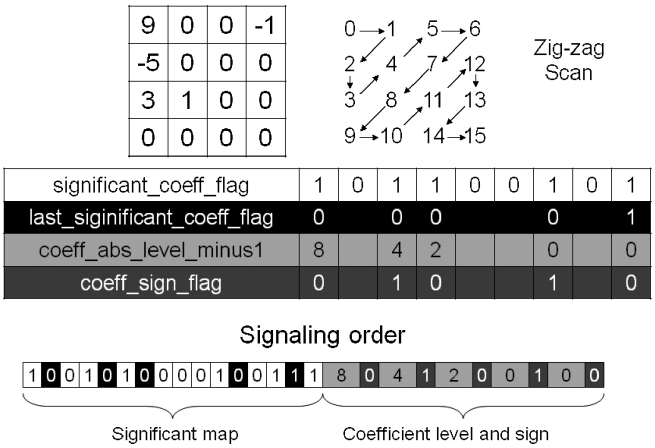

Fig. 3. Example of transform coefficient coding for a 4×4 TU in H.264/AVC.

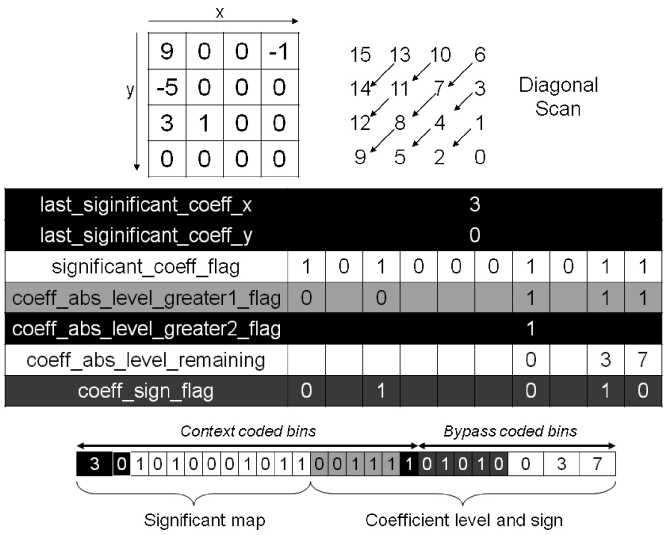

Fig. 4. Example of transform coefficient coding for a 4×4 TU in HEVC (HM-8.0).

SCF, an additional flag called last_significant_coeff_flag (LSCF) is immediately sent to indicate whether it is the last nonzero SCF; this prevents unnecessary SCF from being signaled. Different contexts are used depending on the position within the 4×4 and 8×8 transform unit (TU), and whether the bin represents an SCF or LSCF. Since SCF and LSCF are interleaved, the context selection of the current bin depends on the immediate preceding bin. The dependency of LSCF on SCF results in a strong bin to bin dependency for context selection for significance map in the H.264/AVC as illustrated in Fig. 2.

1) *significant_coeff_flag (SCF):* In HM-1.0, additional dependencies were introduced in the context selection of SCF for 16×16 and 32×32 TU to improve coding efficiency. The context selection for SCF in these larger TU depended on the number of nonzero neighbors to give coding gains between 1.4% to 2.8% [42]. Specifically, the context of SCF depended on up to ten neighbors as shown in Fig. 5(a) [42], [43].

To reduce context selection dependencies, and storage costs, [21] proposed using fewer neighbors and showed that it could be done with minimal cost to coding efficiency. For

TABLE IV
DISTRIBUTION OF BINS IN CABAC FOR H.264/AVC AND HEVC UNDER COMMON CONDITIONS [40], [41] AND WORST CASE

	H.264/AVC			HEVC								
Common condition configurations	HierB	HierP	Worst case	AI MAIN	LP MAIN	LB MAIN	RA MAIN	AI HE10	LP HE10	LB HE10	RA HE10	Worst case
CU bins	27.0%	34.0%	0.5%	4.8%	14.3%	15.1%	10.7%	4.8%	14.2%	15.0%	10.7%	0.6%
PU bins	23.4%	26.3%	15.8%	9.2%	20.6%	19.5%	18.8%	9.1%	20.3%	19.3%	18.7%	5.0%
TU bins	49.7%	39.7%	83.7%	85.4%	63.7%	63.8%	69.4%	85.8%	64.5%	64.6%	69.9%	94.0%
LF bins	n/a	n/a	n/a	0.6%	1.5%	1.6%	1.0%	0.4%	1.1%	1.1%	0.7%	0.8%

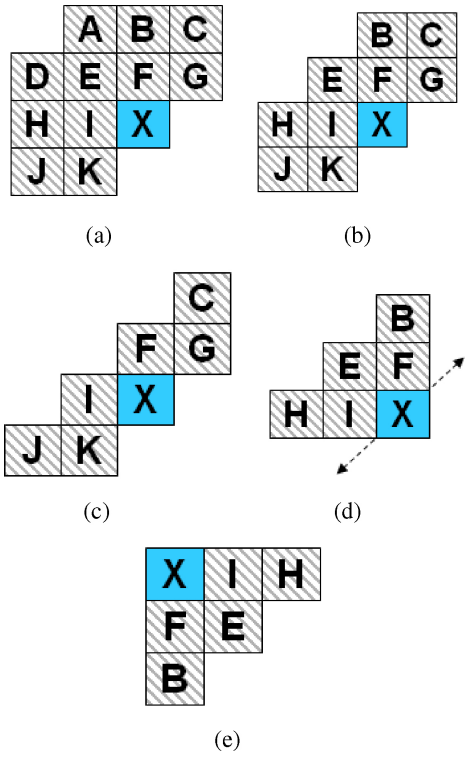

Fig. 5. Neighbor dependencies for SCF context selection. X in blue box represents the current position of the bin being processed. (a) Ten neighbors (HM-1.0). (b) Eight neighbors. (c) Six neighbors. (d) Five neighbors (HM-3.0). (e) Inverted for reverse scan (HM-4.0).

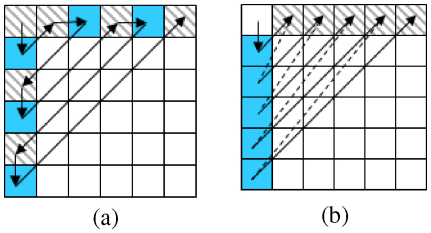

Fig. 6. Scans used to process SCF. Diagonal scan avoids dependency on most recently processed bin. Context selection for blue positions are affected by values of the neighboring gray positions. (a) Zig–zag scan. (b) Diagonal scan.

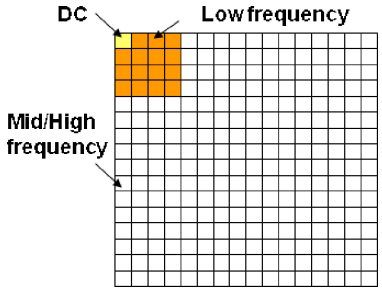

Fig. 7. Regions in 8×8, 16×16, and 32×32 TU map to different context sets for SCF.

instance, using only a maximum of eight neighbors [removing neighbors A and D as shown in Fig. 5(b)] had negligible impact on coding efficiency, while using only six neighbors [removing neighbors A, B, D, E, and H as shown in Fig. 5(c)] results in a coding loss of only 0.2%. This was further extended in [22] for HM-2.0, where only a maximum of five neighbors is used by removing dependencies on positions G and K, as shown in Fig. 5(d). In HM-2.0, 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 SCF.

Despite reducing the neighbors in HM-2.0, dependency on the most recently processed SCF still existed for the positions at the edge of the transform as shown in Fig. 6(a). 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, in HM-4.0, a diagonal scan was introduced to replace the zig–zag scan [44] as shown in Fig. 6(b). Changing from zig–zag to diagonal scan had negligible impact on coding efficiency, but removed dependency on recently processed SCF for all positions in the TU. In HM-4.0, the scan was also reversed (from high frequency to low frequency) [45]. Accordingly, the neighbor dependencies were inverted from top-left to bottom-right, as shown in Fig. 5(e).

Dependencies in the context selection of SCF for 16×16 and 32×32 TU were further reduced in HM-7.0, where 16×16 and 32×32 TU are divided into 4×4 sub-blocks. This will be described in more detail in the section on coded_sub_block_flag (CSBF). In HM-8.0, 8×8 TU was also divided into 4×4 sub-block such that all TU sizes are 4×4 sub-block based for a harmonized design [46].

The 8×8, 16×16, and 32×32 TU are divided into three regions based on frequency, as shown in Fig. 7. The DC, low-frequency, and mid/high-frequency regions all use different sets of contexts. To reduce memory size, the contexts for the SCF of 16×16 and 32×32 are shared [26], [47].

For improved coding efficiency for intra predicted CU, mode-dependent coefficient scanning (MDCS) was introduced which selects between vertical, horizontal, and diagonal scans based on the intra prediction mode [48]. As mentioned in Section V-B, this requires intra mode to be reconstructed

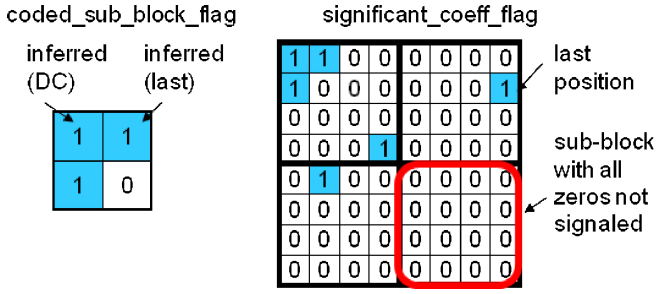

Fig. 8. Example of multilevel signaling of an 8 × 8 significance map.

before decoding coefficients. MDCS is used for 4 × 4 and 8 × 8 TU and provides coding gains of up to 1.2% in intra coded pictures. Note that interpredicted CU only use diagonal scan.

2) *Last Position Coding:* As mentioned earlier, there are strong data dependencies between SCF and LSCF in H.264/AVC due to the fact that they are interleaved. [18] proposed grouping several SCF together by transmitting an LSCF only once per N number of SCF. If all of the N SCF are zero, LSCF is not transmitted. [20] avoids all interleaving of SCF and LSCF altogether. Specifically, the X, Y position of the last nonzero SCF (last_significant_coeff_x and last_significant_coeff_y) is sent rather than LSCF. For instance, in the example shown in Fig. 4, last_significant_coeff_x equal to 3 and last_significant_coeff_y equal to 0 are sent rather than last_significant_coeff_flag. Signaling the X, Y position of the last nonzero SCF was adopted into HM-3.0.

The last position is composed of a prefix and suffix. The prefix (last_significant_coeff_x_prefix, last_significant_coeff_y_prefix) is context coded truncated unary bins with cMax based on width and height of TU for the x and y components, respectively. The suffix (last_significant_coeff_x_suffix, last_significant_coeff_y_suffix) is bypass coded fixed length bins. Some of the contexts are shared across the TU sizes to reduce context memory. To maximize the impact of fast bypass coding, the bypass coded bins (i.e., the suffix) for both the x and y components of the last position are grouped together in HEVC.

3) *coded_sub_block_flag (CSBF):* To reduce the number of bins transmitted for significance map, multilevel signaling is used for the significance map [24], [49]. The TU is divided into 4 × 4 sub-blocks. coded_sub_block_flag is first signaled to indicate whether there are nonzero coefficients in the sub-block; if coded_sub_block_flag is 1, the significant_coeff_flags within the sub-block are signaled. No significant_coeff_flags are signaled for 4 × 4 sub-blocks that contain all zeros. For large TU sizes, up to a 30% reduction in SCF bins is achieved for an overall bin reduction of 3% to 4% under common conditions. To reduce the number of coded_sub_block_flag bins, it is inferred to be one for sub-blocks containing DC and the last position. Fig. 8 shows an example of the multilevel signaling of an 8 × 8 significance map.

In HM-7.0, the CSBF was additionally used to further reduce dependencies in the context selection of SCF for 16×16 and 32×32 TU. Specifically, the neighboring sub-blocks (see Fig. 9) are used for context selection rather than the individual coefficient neighbors of Fig. 5(e) [50]. This was extended to the 8 × 8 TU in HM-8.0 [46]. The CSBF of the neighboring right and bottom sub-blocks ($CSBF_{right}$, $CSBF_{bottom}$) are used to select one of four patterns shown in Fig. 10: (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 sub-block to one of three contexts. Thus, there are no context selection dependencies for the SCF within each 4 × 4 sub-block.

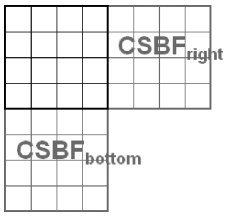

Fig. 9. Neighbor sub-block dependencies for SCF context selection.

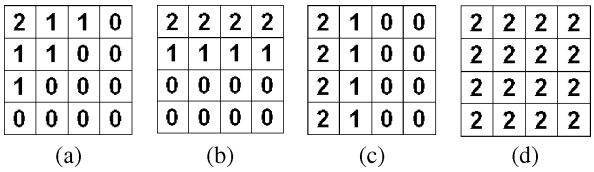

Fig. 10. 4 × 4 position-based mapping for SCF context selection based on SCGF of neighboring sub-blocks. (a) Pattern 1. (b) Pattern 2. (c) Pattern 3. (d) Pattern 4.

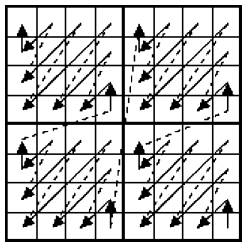

Fig. 11. Sub-block scan.

Diagonal scan is used within the sub-blocks and the sub-blocks themselves are scanned in a diagonal order as shown in Fig. 11 [51]. Both significance map and coefficient level are processed in this order.

To support MDCS in 8×8 TU, vertical, and horizontal scans can also be used within the sub-blocks and the sub-blocks themselves can be scanned in either vertical or horizontal order. Furthermore, different sets of contexts are used for diagonal and non-diagonal (vertical and horizontal) scans [46].

B. Coefficient Level and Sign

In H.264/AVC, the coefficient level is composed of two parts. The first 14 bins, generated with truncated unary binarization, are context coded. The remaining bins, generated by 0th-order Exp-Golomb binarization, are bypass coded bins. After each coefficient level is signaled, the sign is signaled with one bypass bin.

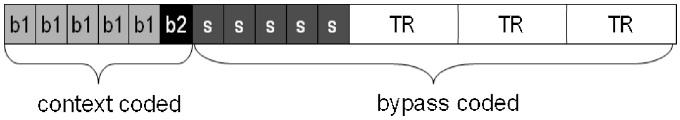

Fig. 12. Grouping same context coded bins and bypass bins to increase throughput. b1=coeff_abs_level_greater1_flag, b2=coeff_abs_level_greater2_flag, s=coeff_sign_flag, TR=coeff_abs_level_remaining.

1) *Level Coding:* In HEVC, only the first two bins of the coefficient level (coeff_abs_level_greater1_flag and coeff_abs_level_greater2_flag) are context coded [12]. The remaining portion of the levels (coeff_abs_level_remaining) is bypass coded. Thus, the maximum number of context coded bins in coefficient level is reduced by 7×. This change provides the most substantial reduction to the number of context coded bins in the entire CABAC.

coeff_abs_level_remaining is binarized using unary coding for the prefix, and fixed length coding for the suffix; the number of fixed length bins depends on the prefix value [8]. To improve coding efficiency, the number of fixed length bins also depends on parameter cRiceParam, which adaptively changes based on the value of previous nonzero coefficient level. This introduces a dependency between coefficient levels, but does not impact throughput unless more than one level is coded at the same time. The binarization is designed such that the maximum bin length of coeff_abs_level_remaining is 32 [23].

In HEVC, bins in the coefficient level that use the same context selection logic are grouped together to reduce the amount of speculative context selection computations as shown in Fig. 12. Specifically, coeff_abs_level_greater1_flag are grouped together followed by coeff_abs_level_greater2_flag [15], [19]. The bypass coded bins are grouped together, which maximizes the throughput advantages of bypass bins [15]. This reordering of bins to enable PCP has no impact on coding efficiency.

To further reduce the number of context coded bins, only a maximum of eight coeff_abs_level_greater1_flag and one coeff_abs_level_greater2_flag are sent per 4 × 4 sub-block [14]. Accordingly, in the example shown in Fig. 4, coeff_abs_level_greater2_flag is not signaled for coefficients −5 and 9 since it is already signaled once for coefficient 3; the coefficient level for coefficient −5 and 9 are reconstructed by adding coeff_abs_level_greater1_flag and coeff_abs_level_remaining. Limiting the number of coeff_abs_level_greater1_flag and coeff_abs_level_greater2_flag per 4 × 4 sub-block also helps to reduce the number of contexts.

Reducing the total number of contexts not only lowers storage cost, but can also reduce context read time which can help increase throughput (assuming that the complexity of context selection logic does not increase). Thus, several steps were taken to reduce the number of contexts used for level coding.

Context selection for coefficient level involves the following steps.

1) Sixteen successive coefficients in scanning order form a sub-block.
2) A context set is assigned to the sub-block based on the number of coefficients greater than one in previous sub-blocks.
3) A context is selected from within that set based the number of trailing ones in the sub-block.

In HM-3.0, each context set contained five contexts. There were a total of six context sets: three for low frequency, and three for the rest. coeff_abs_level_greater1_flag and coeff_abs_level_greater2_flag have different contexts. Luma and chroma also have different contexts. Thus, level coding required 120 context calculated as follows:

$$5 \text{ (number of contexts per set)} \times 6 \text{ (number of sets)} \times 2 \text{ (number of syntax elements)} \times 2 \text{ (luma/chroma)} = \mathbf{120}. \quad (1)$$

For HM-4.0, it was shown in [52] that 36 contexts can be removed by reducing the number of contexts per set from 5 to 4 for coeff_abs_level_greater1_flag, and 5 to 3 for coeff_abs_level_greater2_flag, with negligible impact on coding efficiency. In HM-5.0, the number of context sets for chroma was reduced from 6 to 2 [53]. Finally, in HM-6.0, the number of contexts sets for luma was reduced from 6 to 4, and the number of contexts per context set for coeff_abs_level_greater2_flag was further reduced, since only a single flag is coded per 16 coefficients. Thus, coeff_abs_level_greater1_flag has 24 contexts as shown in

$$4 \text{ (number of contexts per set)} \times (\ 2 \text{ (chroma sets)} + 4 \text{ (luma sets)}\ ) = \mathbf{24\ contexts} \quad (2)$$

and coeff_abs_level_greater2_flag has 6 contexts, as shown in

$$1 \text{ (number of contexts per set)} \times (\ 2 \text{ (chroma sets)} + 4 \text{ (luma sets)}\ ) = \mathbf{6\ contexts}. \quad (3)$$

From HM-3.0 to HM-6.0, there was a 4× reduction (from 120 to 30) in the total number of contexts used for coefficient levels.

2) *Sign:* To reduce storage cost of the coefficients, the data is grouped for every 4 × 4 sub-block (16 coefficients) and the sign bins are 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-b. Thus, the CABAC only requires storage of 4 × 4×4-b (as compared to 8×8×9-b in H.264/AVC), and the reconstructed coefficient level can be immediately written out once coeff_abs_level_remaining is parsed.

To improve coding efficiency, data hiding is used such that the sign flag for the first nonzero coefficient is not always sent [54]. If the number of nonzero coefficients in a sub-block exceeds a certain threshold, then the sign flag can be inferred from whether the sum of the coefficients in the sub-block is even or odd. The condition for sign data hiding can be checked while parsing the significance map and thus does not have a significant impact on the entropy decoding throughput. Coding efficiency improvement between 0.6 to 1.4% is achieved [55].

C. Comparison of H.264/AVC and HEVC

Table V summarizes the differences in TU coding between H.264/AVC and HEVC and differences across TU sizes. In

TABLE V
DIFFERENCES BETWEEN TRANSFORM COEFFICIENT CODING FOR DIFFERENT TU SIZES IN HEVC AND H.264/AVC
[CONTEXTS FOR H.264/AVC INCLUDES INTERLACED (×2)]

Properties		**HEVC**				**H.264/AVC**	
TU size		**4×4**	**8×8**	**16×16**	**32×32**	**4×4**	**8×8**
Context selection for 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
	significant_coeff_flag	8 (Y), 8 (CbCr)	12 (Y), 3 (CbCr)	6 (Y), 3 (CbCr)		44×2 (Y), 17×2 (CbCr)	15×2 (Y)
		1 (Y), 1 (CbCr), DC context shared across TU sizes					
	last_significant_coeff_x, y (HEVC) or last_significant_coeff_flag (H.264/AVC)	15×2 (Y), 3×2 (CbCr)				44×2 (Y) 17×2 (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 context coded bins per TU	coded_sub_block_flag	0	2	14	62	n/a	n/a
	significant_coeff_flag	15	63	255	1023	15	63
	last_significant_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 context coded bins per coefficient		1.9	1.7	1.7	1.6	17.1	16.0

terms of the impact of the throughput improvement techniques, HEVC requires 3× fewer contexts (124 versus 398) than H.264/AVC for TU coding. Furthermore, HEVC has 9× fewer context coded bins per coefficient (1.9 versus 17.1) than H.264/AVC.

Table VI summarizes the coding efficiency impact of the various adopted tools. The majority of the tools adopted focused on throughput improvements with minimal coding loss.

TABLE VI
CODING EFFICIENCY IMPACT OF ADOPTED TU CODING TOOLS (CODING EFFICIENCIES ARE LISTED IN BD-RATE [56], WHERE POSITIVE VALUES INDICATE CODING LOSS AND NEGATIVE VALUES INDICATE CODING GAIN)

Tool	**HM**	**Benefit**	**BD-rate**
Neighbor-based context selection for SCF [43]	1.0	Coding gain	−2.8% to −1.4%
Group bypass sign [19]	1.0	Throughput	0.0%
Mode-dependent coefficient scanning [48]	2.0	Coding gain	−1.2% to −0.1%
Reduce neighboring dependency for SCF [22]	2.0	Throughput	−0.1% to 0.0%
Reduce context coded level bins [12]	3.0	Throughput	−0.1% to 0.0%
Last position coding [20]	3.0	Throughput	−0.1% to 0.0%
Group bypass level [15]	4.0	Throughput	0.0%
Diagonal scan [44]	4.0	Throughput	−0.1% to 0.0%
SCGF and sub-block scan [24], [51]	5.0	Throughput	−0.1% to 0.1%
Reduce number of context coded level bins per 4 × 4 [14]	6.0	Throughput	−0.1% to 0.1%
Sign data hiding [55]	6.0	Coding gain	−1.4% to −0.6%
Use SCGF of neighboring sub-blocks for SCF [50]	7.0	Throughput	0.1% to 0.2%

VII. OVERALL PERFORMANCE

This section describes the overall improvements to the CABAC in HEVC test model at Draft International Standard (HM-8.0 [6]) compared with the CABAC in H.264/AVC [57]. Simulations were performed under common conditions set by the JCT-VC [40], [41]. Common conditions are intended to reflect the typical bitstream and is the configuration used under which coding efficiency of proposals are evaluated. Analysis was also done for the worst case throughput which is defined as the case with the most number of bins per LCU or macroblock (i.e., 16 × 16 block of pixels). The results for both common conditions and worst case are summarized in Tables VII and VIII, respectively.

A. Throughput Analysis

This section describes throughput of HEVC relative to H.264/AVC. The impact of the techniques, outlined in Section IV, are discussed.

1) *Reduce Context Coded Bins:* As mentioned earlier, bypass coded bins can be processed faster than context coded bins, since they do not have data dependencies due to context selection, and their range division can be performed by a simple shift. Table VII shows that the percentage of context coded bins under common conditions is lower for HEVC than H.264/AVC. Table VIII also shows that in the worst case conditions, there are 8× fewer context coded bins in HEVC than H.264/AVC. The reduction in context coded bins is primarily attributed to the modifications to coefficient level and motion vector difference.

TABLE VII
DISTRIBUTION OF CONTEXT CODED, BYPASS, AND TERMINATION BINS FOR CABAC IN H.264/AVC AND HEVC UNDER COMMON CONDITIONS [41], [58]

	Common condition configurations	Context (%)	Bypass (%)	Term (%)
H.264/AVC	HierB	80.5	13.6	5.9
	HierP	79.4	12.2	8.4
HEVC	AI_MAIN	67.9	32.0	0.1
	LP_MAIN	78.2	20.8	1.0
	LB_MAIN	78.2	20.8	1.0
	RA_MAIN	73.0	26.4	0.6
	AI_HE10	68.1	31.8	0.1
	LP_HE10	78.6	20.4	1.0
	LB_HE10	78.6	20.4	1.0
	RA_HE10	73.3	26.1	0.6

TABLE VIII
WORST CASE BIN AND MEMORY REDUCTION IN HEVC OVER H.264/AVC

Metric	H.264/AVC	HEVC	Reduction
Max context coded bins	7805	939	8×
Max bypass bins	13 056	13 425	1×
Max total bins	20 861	14 364	1.5×
Number of contexts	447	154	3×
Line buffer for 4k×2k	30 720	1536	20×
Coefficient storage	8×8×9-b	4×4×4-b	9×
Initialization table	64 032	3536	18×

Using the implementation found in [58], where up to two context coded bins or four 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.

2) *Group Bypass Coded Bins:* Grouping bypass bins together can increase the number of bins processed per cycle and reduced the number of cycles required to process the bypass bins. This is a technique used on motion vector difference, intra mode, last position, and coefficient levels. In HEVC, under common conditions, grouping of bypass bins results in up to a 2.8× reduction in number of bypass cycles relative to not grouping.

The benefit of bypass grouping can also be seen in the example of Figs. 3 and 4. If bypass grouping was not used, it would take five cycles to process the five sign bypass bins. Assuming the architecture of [58], where four bypass bins are processed per cycle, only two cycles are required to process the five sign bins.

3) *Group Bins With the 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 rules for context selection are grouped together.

Fig. 2 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. 13. Thus, for this example, the number of operations are reduced from 14 to 5.

TABLE IX
CONTEXT MEMORY REQUIREMENTS FOR H.264/AVC (4:2:0) AND HEVC

	H.264/AVC		HEVC
	(w/ interlace)	(w/o interlace)	
CU contexts	23	20	14
PU contexts	26	26	14
TU contexts	398	252	124
LF contexts	n/a	n/a	2
Total	**447**	**298**	**154**

4) *Reduce Context Selection Dependencies:* Context selection dependencies were reduced such that coding gains could be achieved without significant penalty to throughput. For instance, last significant coefficient position information is sent before significant_coeff_flag to remove a tight bin to bin data dependency. Relative to HM-1.0, the neighboring dependencies for significant_coeff_flag were reduced from 10 to 5, and then further modified to only depend on neighboring 4×4 sub-blocks. The rest of the significant_coeff_flag are position based as in H.264/AVC.

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 are required per bin, 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.

6) *Reduce Parsing Dependencies:* Parsing dependencies were removed or reduced such that coding gains could be achieve without significant penalty to throughput. Removing the parsing dependency for merge and mvp enables parsing to be mostly decoupled from other video modules as in H.264/AVC. HEVC does have parsing dependencies on intra mode reconstruction, which is not present in H.264/AVC; however, effort was made to keep intra mode reconstruction simple to avoid effecting parsing throughput.

B. Memory Requirement Reduction

This section describes how the various memories in CABAC have been reduced in HEVC.

1) *Context Memory:* The motivation for context reduction was first proposed in [26], 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 [59]–[61] were made to reduce the number of contexts for other syntax elements (e.g., significant_coeff_flag). HEVC uses only 154 contexts as compared to 447 (or 298 without interlaced) used in H.264/AVC as shown in Table IX; thus, a 3× reduction in context memory size is achieved with HEVC.

2) *Line Buffer Memory:* The motivation to reduce the size of the line buffer in the CABAC was first proposed in [27], [62], where the line buffer size was reduced by changing the context selection for motion vector difference. Subsequent proposals [13], [33], [34], [63]–[65] were made to further reduce neighboring dependencies to reduce the line buffer size.

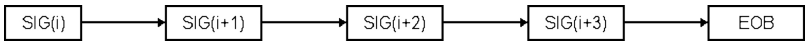

Fig. 13. No context speculation is required to achieve 5× parallelism when processing the 4 × 4 significance map in HEVC. i = coefficient position. EOB = end of block. SIG = significant_coeff_flag.

TABLE X
SUMMARY OF THROUGHPUT IMPROVEMENT TECHNIQUES AND RELATED CONTRIBUTIONS

Technique	PU coding	TU coding
Reduce context coded bins	[13]	[12], [14]
Group bypass bins	[16], [17]	[15]
Group bins with same context	[13]	[18]–[20]
Reduce context selection dependencies		[11], [21] [22], [44]
Reduce total number of bins		[23], [24]
Reduce memory requirements	[13], [27] [33], [34]	[26], [34], [47], [52] [53], [59]–[61], [63]–[65]
Reduce parsing dependencies	[25], [31]	

Based on these optimizations, only 3-b are required per CU in the line buffer to store the CU depth (2-b) of the top neighbor for context selection of split_coding_unit_flag, and to indicate if the top neighbor is skipped (1-b) for the context selection of skip_flag. Assuming a minimum CU size of 8×8 for a $4k\times2k$ sequence, HEVC only requires a line buffer size of 1536-b versus 30 720-b in H.264/AVC, which is a 20× reduction.

3) *Coefficient Storage:* Large transform sizes have large hardware cost implications. Compared to H.264/AVC, the 16×16 and 32×32 transforms in HEVC have 4× and 16× more coefficients and consequently requires 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 4-b storage is required per coefficient for the partial decoded value. Second, the coefficient information is interleaved at a 16 coefficient level, such that the fully constructed coefficient can be achieved every 16 and be sent out to the next module [66]. Thus, only a coefficient storage of 4×4×4-b is required in HEVC CABAC (compared with 8×8×9-b in H.264/AVC) in order to reconstruct the coefficient levels.

4) *Context Initialization Tables:* To reduce storage table sizes for context initialization, [67] first proposed to use 8-b values to derive the initial context state, rather than the 16-b values used in H.264/AVC. In both H.264/AVC and HEVC, the initial state and MPS for each context is derived using a linear model. The 8-b initValues are mapped to the slope (m) and offset (n) of this model using [68]

$$m = \text{initValue}[7{:}4]\times5 - 45$$
$$n = (\text{initValue}[3{:}0] \ll 3) - 16.$$

In H.264/AVC, there are different sets of initValue depending on slice type and for each slice type there are three sets of initValue depending on cabac_init_idc. In HEVC, each slice type has one set of initValue and cabac_init_idc is replaced by cabac_init_flag which allows a P slice to be initialized by the initValue of a B slice and vice versa [69]. Accounting for the reduction in number of contexts, number of bits per initValue and number of initValue sets, HEVC has 18× smaller context initialization table than H.264/AVC.

VIII. CONCLUSION

Entropy coding was a highly active area of development throughout the HEVC standardization process with proposals for both coding efficiency and throughput improvement. The tradeoff between the two requirements was carefully evaluated in multiple core experiments and *ad hoc* groups [39], [70]–[73]. Many techniques were used to improve throughput, including reducing context 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. A summary of the throughput techniques and related contributions are found in Table X. The final design shows that accounting for implementation cost when designing video coding algorithms results in a design that can maximize processing speed and minimize area cost, while delivering high coding efficiency in the next generation video coding standard.

ACKNOWLEDGMENT

The work presented in this paper was carried out as a part of several core experiments and *ad hoc* groups on entropy coding and coefficient scanning and coding for HEVC standardization [39], [70]–[73].

REFERENCES

[1] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Trans. Circuits Syst. Video Technol.,* vol. 13, no. 7, pp. 620–636, Jul. 2003.

[2] *Recommendation ITU-T H.264: Advanced Video Coding for Generic Audiovisual Services*, Tech. Rep., ITU-T, 2003.

[3] B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan, and T. Wiegand, *High Efficiency Video Coding (HEVC) Text Specification Draft 8*, document JCTVC-J1003, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2012.

[4] V. Sze, M. Budagavi, and M. U. Demircin, *CABAC Throughput Requirements for Real-Time Decoding*, document VCEG-AJ31, Video Coding Experts Group (VCEG), Oct. 2008.

[5] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuit,* vol. 27, no. 4, pp. 473–484, Apr. 1992.

[6] *HEVC Test Model, HM 8.0*. (2012, Aug.) [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-8.0/

[7] T. Davies and A. Fuldseth, *Entropy Coding Performance Simulations*, document JCTVC-F162, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[8] W.-J. Chien, M. Karczewicz, J. Sole, and J. Chen, *On Coefficient Level Remaining Coding*, document JCTVC-I0487, Joint Collaborative Team on Video Coding (JCT-VC), Apr. 2012.

[9] V. Sze, M. Budagavi, V. Seregin, J. Sole, and M. Karczewicz, *AHG5: Bin Reduction for Delta QP Coding*, document JCTVC-J0089, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2012.

[10] V. Sze, M. Budagavi, A. Chandrakasan, and M. Zhou, "Parallel CABAC for low power video coding," in *Proc. IEEE Int. Conf. ICIP*, Oct. 2008, pp. 2096–2099.

[11] V. Sze, *Context Selection Complexity in HEVC CABAC*, document JCTVC-D244, Joint Collaborative Team on Video Coding (JCT-VC), Jan. 2011.

[12] T. Nguyen, *CE11: Coding of Transform Coefficient Levels With Golomb-Rice Codes*, document JCTVC-E253, Joint Collaborative Team on Video Coding (JCT-VC), Mar. 2011.

[13] T. Nguyen, D. Marpe, H. Schwarz, and T. Wiegand, *Modified Binarization and Coding of MVD for PIPE/CABAC*, document JCTVC-F455, Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.

[14] J. Chen, W. J. Chien, R. Joshi, J. Sole, and M. Karczewicz, *Non-CE1: Throughput Improvement on CABAC Coefficients Level Coding*, document JCTVC-H0554, Joint Collaborative Team on Video Coding (JCT-VC), Feb. 2012.

[15] V. Sze and M. Budagavi, *Parallel Context Processing of Coefficient Level*, document JCTVC-F130, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[16] H. Sasai and T. Nishi, *Modified MVD Coding for CABAC*, document JCTVC-F423, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[17] W.-J. Chien, J. Chen, M. Coban, and M. Karczewicz, *Intra Mode Coding for INTRA_NxN,* document JCTVC-I0302, Joint Collaborative Team on Video Coding (JCT-VC), Apr. 2012.

[18] M. Budagavi and M. U. Demircin, *Parallel Context Processing Techniques for High Coding Efficiency Entropy Coding in HEVC*, document JCTVC-B088, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2010.

[19] M. Budagavi, *TE8: TI Parallel Context Processing (PCP) Proposal*, document JCTVC-C062, Joint Collaborative Team on Video Coding (JCT-VC), Oct. 2010.

[20] J. Sole, R. Joshi, and M. Karczewicz, *CE11: Parallel Context Processing for the Significance Map in High Coding Efficiency*, document JCTVC-E338, Joint Collaborative Team on Video Coding (JCT-VC), Mar. 2011.

[21] V. Sze and M. Budagavi, *Parallelization of HHI_TRANSFORM_CODING*, document JCTVC-C227, Joint Collaborative Team on Video Coding (JCT-VC), Oct. 2010.

[22] A. Cheung and W. Lui, *Parallel Processing Friendly Simplified Context Selection of Significance Map*, document JCTVC-D260, Joint Collaborative Team on Video Coding (JCT-VC), Jan. 2011.

[23] M. Budagavi and V. Sze, *coeff_abs_level_remaining Maximum Codeword Length Reduction,* document JCTVC-J0142, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2012.

[24] N. Nguyen, T. Ji, D. He, G. Martin-Cocher, and L. Song, *Multi-Level Significant Maps for Large Transform Units*, document JCTVC-G644, Joint Collaborative Team on Video Coding (JCT-VC), Nov. 2011.

[25] M. Zhou, V. Sze, and Y. Mastuba, *A Study on HEVC Parsing Throughput Issue*, document JCTVC-F068, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[26] V. Sze, *Reduction in Contexts Used for significant_coeff_flag and Coefficient Level*, document JCTVC-F132, Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.

[27] V. Sze and A. P. Chandrakasan, *Joint Algorithm-Architecture Optimization of CABAC*, document JCTVC-E324, Joint Collaborative Team on Video Coding (JCT-VC), Mar. 2011.

[28] B. Bross and J. Jung, *Description of Core Experiment CE13: Motion Data Parsing Robustness and Throughput*, document JCTVC-F913, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[29] T. Hellman and Y. Yu, *Decoder Performance Restrictions due to Merge/MVP Index Parsing*, document JCTVC-F341, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[30] T. Sugio and T. Nishi, *Parsing Robustness for Merge/AMVP*, document JCTVC-F470, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[31] M. Zhou and V. Sze, *A Study on HM2.0 Bitstream Parsing and Error Resiliency Issue*, document JCTVC-E118, Joint Collaborative Team on Video Coding (JCT-VC), Mar. 2011.

[32] J. Jung and G. Laroche, *Competition-Based Scheme for Motion Vector Selection and Coding*, document VCEG-AC06, Video Coding Experts Group (VCEG), Jul. 2006.

[33] V. Sze and A. P. Chandrakasan, *Simplified MVD Context Selection (Extension of JCTVC-E324)*, document JCTVC-F133, Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.

[34] V. Sze, *BoG Report on Context Reduction for CABAC*, document JCTVC-F746, Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.

[35] V. Sze and R. Allen, *BoG Report on Intra Mode Coding*, document JCTVC-G1017, Joint Collaborative Team on Video Coding (JCT-VC), Nov. 2011.

[36] K. Chono, *BoG Report on Intra Mode Coding Cleanup and Simplification*, document JCTVC-H0712, Joint Collaborative Team on Video Coding (JCT-VC), Feb. 2012.

[37] C. Lan, J. Xu, G. J. Sullivan, and F. Wu, *Intra Transform Skipping*, document JCTVC-I0408, Joint Collaborative Team on Video Coding (JCT-VC), Apr. 2012.

[38] X. Peng, C. Lan, J. Xu, and G. J. Sullivan, *Inter Transform Skipping*, document JCTVC-J0237, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2012.

[39] V. Sze, K. Panusopone, J. Chen, T. Nguyen, and M. Coban, *Description of Core Experiment 11: Coefficient Scanning and Coding*, document JCTVC-C511, Joint Collaborative Team on Video Coding (JCT-VC), Oct. 2010.

[40] *Joint Call for Proposals on Video Compression Technology*, document VCEG-AM91, ITU-T Q6/16 Visual Coding and ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Audio, Jan. 2010.

[41] F. Bossen, *HM 8 Common Test Conditions and Software Reference Configurations*, document JCTVC-J1100, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2012.

[42] T. Nguyen, D. Marpe, H. Schwarz, and T. Wiegand, *CE11: Evaluation of Transform Coding Tools in HE Configuration*, document JCTVC-D061, Joint Collaborative Team on Video Coding (JCT-VC), Jan. 2011.

[43] M. Winken, S. BoBe, B. Bross, P. Helle, T. Hinz, H. Kirchhoffer, H. Lakshman, D. Marpe, S. Oudin, M. PreiB, H. Schwarz, M. Siekmann, K. Suhring, and T. Wiegand, *Description of Video Coding Technology Proposal by Fraunhofer HHI*, document JCTVC-A116, Joint Collaborative Team on Video Coding (JCT-VC), Apr. 2010.

[44] V. Sze and M. Budagavi, *CE11: Parallelization of HHI_TRANSFORM_CODING Fixed Diagonal Scan*, document JCTVC-F129, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[45] J. Sole, R. Joshi, and M. Karczewicz, *CE11: Unified Scans for the Significance Map and Coefficient Level Coding in High Efficiency*, document JCTVC-F288, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[46] J. Sole, R. Joshi, and M. Karczewicz, *Removal of the* $8 \times 2/2 \times 8$ *Coefficient Groups*, document JCTVC-J0256, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2012.

[47] K. Terada, H. Sasai, and T. Nishi, *Non-CE11: Simplification of Context Selection for significant_coeff_flag*, document JCTVC-H0290, Joint Collaborative Team on Video Coding (JCT-VC), Feb. 2012.

[48] Y. Zheng, M. Coban, X. Wang, J. Sole, R. Joshi, and M. Karczewicz, *CE11: Mode Dependent Coefficient Scanning*, document JCTVC-D393, Joint Collaborative Team on Video Coding (JCT-VC), Jan. 2011.

[49] N. Nguyen, T. Ji, D. He, and G. Martin-Cocher, *Non-CE1: Throughput Improvement on CABAC Coefficients Level Coding*, document JCTVC-H0554, Joint Collaborative Team on Video Coding (JCT-VC), Feb. 2012.

[50] S. Kumakura, T. Fukushima, *Non-CE3: Simplified Context Derivation for Significance Map*, document JCTVC-I0296, Joint Collaborative Team on Video Coding (JCT-VC), Apr. 2012.

[51] J. Sole, R. Joshi, and M. Karczewicz, *Non-CE11: Diagonal Sub-Block Scan for HE Residual Coding*, document JCTVC-G323, Joint Collaborative Team on Video Coding (JCT-VC), Nov. 2011.

[52] V. Sze, *Reduction in Contexts Used for significant_coeff_flag and Coefficient Level*, document JCTVC-F132, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[53] Y. Piao, J. Min, E. Alshina, and J. T. Park, *Reduced Chroma Contexts for Significance Map Coding in CABAC*, document JCTVC-G781, Joint Collaborative Team on Video Coding (JCT-VC), Nov. 2011.

[54] G. Clare, F. Henry, and J. Jung, *Sign Data Hiding*, document JCTVC-G271, Joint Collaborative Team on Video Coding (JCT-VC), Nov. 2011.

[55] X. Yu, J. Wang, D. He, G. Martin-Cocher, and S. Campbell, *Multiple Sign Bits Hiding*, document JCTVC-H0481, Joint Collaborative Team on Video Coding (JCT-VC), Feb. 2012.

[56] G. Bjontegaard, *VCEG-M33: Calculation of Average PSNR Differences Between RD Curves*, Video Coding Experts Group (VCEG), Apr. 2001.

[57] *H.264/AVC Reference Software, JM 16.2* [Online]. Available: http://iphome.hhi.de/suehring/tml/

[58] Y. C. Yang and J. I. Guo, "High-throughput H.264/AVC high-profile CABAC decoder for HDTV applications," *IEEE Trans. Circuit Syst. Vedio Technol.,* vol. 19, no. 9, pp. 1395–1399, Sep. 2009.

[59] H. Sasai and T. Nishi, *CE11: Context Size Reduction for the Significance Map*, document JCTVC-E227, Joint Collaborative Team on Video Coding (JCT-VC), Mar. 2011.
[60] V. Sze and H. Sasai, *Modification to JCTVC-E227 in CE11 for Reduced Dependency With MDCS*, document JCTVC-E489, Joint Collaborative Team on Video Coding (JCT-VC), Mar. 2011.
[61] C. Auyeung, J. Xu, G. Korodi, J. Zan, D. He, Y. Piao, J. Alshina, E. Min, and J. Park, *A Combined Proposal from JCTVC-G366, JCTVC-G657, and JCTVC-G768 on Context Reduction of Significance Map Coding With CABAC*, document JCTVC-G1015, Joint Collaborative Team on Video Coding (JCT-VC), Nov. 2011.
[62] V. Sze and A. P. Chandrakasan, "Joint algorithm-architecture optimization of CABAC to increase speed and reduce area cost," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, May 2011, pp. 1577–1580.
[63] H. Sasai and T. Nishi, *Modified Context Derivation for Neighboring Dependency Reduction*, document JCTVC-F429, Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.
[64] V. Seregin and I. K. Kim, *Binarisation Modification for Last Position Coding*, document JCTVC-F375, Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.
[65] W.-J. Chien, M. Karczewicz, and X. Wang. (2009, Nov.). *Memory and Parsing Friendly CABAC Context*, document JCTVC-F606, Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.
[66] J. Sole, R. Joshi, and M. Karczewicz, *Scanning of Residual Data in HE*, document JCTVC-F552:CE11, Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.
[67] D. Marpe, H. Kirchhoffer, B. Bross, V. George, T. Nguyen, M. Preiß, M. Siekmann, J. Stegemann, and T. Wiegand, *Unified PIPE-Based Entropy Coding for HEVC*, document JCTVC-F268, Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.
[68] L. Guo, J. Sole, R. Joshi, C. Karczewicz, M. Yeo, Y. Tan, and Z. Li, *CE1-B3: 8-Bit Linear Initialization for CABAC*, document JCTVC-H0535, Joint Collaborative Team on Video Coding (JCT-VC), Feb. 2012.
[69] K. Misra and A. Segall, *CE1: Subtest B4—On cabac_init_idc*, document JCTVC-H0540, Joint Collaborative Team on Video Coding (JCT-VC), Feb. 2012.
[70] M. Budagavi and A. Segall, *AHG Report: Parallel Entropy Coding*, document JCTVC-B009, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2010.
[71] M. Budagavi, *Tool Experiment 8: Parallel Entropy Coding*, document JCTVC-B308, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2010.
[72] M. Budagavi, G. Martin-Cocher, and A. Segall, *JCT-VC AHG Report: Entropy Coding*, document JCTVC-D009, Joint Collaborative Team on Video Coding (JCT-VC), Mar. 2010.
[73] R. Joshi, E. Alshina, H. Sasai, H. Kirchhoffer, and J. Lainema, *Description of Core Experiment 1: Entropy Coding*, document JCTVC-F901, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

Vivienne Sze (M'10) received the B.A.Sc. (Hons.) degree in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 2004, and the S.M. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 2006 and 2010, respectively.

Since 2010, she has been a Technical Staff Member with the Systems and Applications Research and Development Center, Texas Instruments (TI), Dallas, where she designs low-power algorithms and architectures for video coding. She also represents TI at the international JCT-VC standardization body developing high-efficiency video coding, the next-generation video coding standard. Within the committee, she is the Primary Coordinator of the core experiment on coefficient scanning and coding, and has been the Chair and Vice Chair of several *ad hoc* groups related to entropy coding.

Dr. Sze was a recipient of the 2007 DAC/ISSCC Student Design Contest Award and a co-recipient of the 2008 A-SSCC Outstanding Design Award. She received the Jin-Au Kong Outstanding Doctoral Thesis Prize, awarded for the Best Ph.D. Thesis in Electrical Engineering in 2011. She received the Natural Sciences and Engineering Research Council of Canada (NSERC) Julie Payette Fellowship in 2004, the NSERC Postgraduate Scholarship in 2005 and 2007, and the Texas Instruments Graduate Woman's Fellowship for Leadership in Microelectronics in 2008.

Madhukar Budagavi (SM'05) received the B.E. (First Class With Distinction) degree in electronics and communications engineering from the National Institute of Technology, Trichy, India, in 1991, the M.Sc. (Eng.) degree in electrical engineering from the Indian Institute of Science Bangalore, India, in 1994, and the Ph.D. degree in electrical engineering from Texas A&M University, College Station, in 1998.

From 1993 to 1995, he was with Motorola India Electronics, Ltd., Bangalore, developing DSP software and algorithms for Motorola DSP chips. Since 1998, he has been with the Texas Instruments (TI) Systems and the Application Research and Development Center, engaged in video coding, 3-D graphics, and image processing research, design, and implementation. From 2003 to 2007, he was an Adjunct Assistant Professor with Southern Methodist University, University Park, TX, where he taught courses in DSP and image processing to undergraduate and graduate students. He has published more than 35 journal and conference papers (including seven book chapters) in the field of video coding, multimedia communications, DSP programming, speech coding, and biomedical data compression. He has been representing TI in ITU and ISO international video coding standardization activity. His most recent participation has been in the next-generation video coding standard HEVC being standardized by the JCTVC Committee of ITU and ISO. Within the committee, he has helped coordinate core experiments and AhG activity on spatial transforms, quantization, entropy coding, in-loop filters, and intra prediction.