A Highly Parallel and Scalable CABAC Decoder for Next Generation Video Coding

Vivienne Sze, *Member, IEEE*, and Anantha P. Chandrakasan, *Fellow, IEEE*

***Abstract*—Future video decoders will need to support high resolutions such as Quad Full HD (QFHD, 4096 $\times$ 2160) and fast frame rates (e.g., 120 fps). Many of these decoders will also reside in portable devices. Parallel processing can be used to increase the throughput for higher performance (i.e., processing speed), which can be traded-off for lower power with voltage scaling. The next generation standard called High Efficiency Video Coding (HEVC), which is being developed as a successor to H.264/AVC, not only seeks to improve the coding efficiency but also to account for implementation complexity and leverage parallelism to meet future power and performance demands. This paper presents a silicon prototype for a pre-standard algorithm developed for HEVC ("H.265") called Massively Parallel CABAC (MP-CABAC) that addresses a key video decoder bottleneck. A scalable test chip is implemented in 65-nm and achieves a throughput of 24.11 bins/cycle, which enables it to decode the max H.264/AVC bit-rate (300 Mb/s) with only a 18 MHz clock at 0.7 V, while consuming 12.3 pJ/bin. At 1.0 V, it decodes a peak of 3026 Mbins/s for a bit-rate of 2.3 Gb/s, enough for QFHD at 186 fps. Both architecture and joint algorithm-architecture optimizations used to reduce critical path delay, area cost and memory size are discussed.**

***Index Terms*—CABAC, CMOS digital integrated circuits, entropy coding, HEVC, H.264/AVC, low-power electronics, parallel algorithms, parallel architectures, video codecs, video coding.**

I. Introduction

TODAY'S video codecs have both power and performance requirements. High performance (i.e., processing speed) is needed to deliver the target resolutions and frame rates and low power consumption is needed to extend battery life. Scalability is also desirable, such that a single video codec can support a wide variety of applications. Next-generation video codecs will be expected to achieve at least 4k $\times$ 2k Quad Full High Definition (QFHD) resolution for ultra-high definition, which has 4$\times$ the number of pixels per frame compared to today's 1080 high definition; frame rates are also expected to increase to 120 frames per second (fps) and beyond to support high-motion sequences and slow-motion playback. As a result, over an order of magnitude increase in data is expected compared to today's high definition, which means $>10\times$ processing speed and better compression is required.

The Joint Collaborative Team on Video Coding (JCT-VC) is currently developing the next generation video coding standard called High Efficiency Video Coding (HEVC) [1]. It is expected to deliver 50% better coding efficiency than its predecessor, H.264/AVC, which is today's state-of-the-art video coding standard. This improvement in coding efficiency is likely to come at a cost of increased complexity and thus power reduction will continue to be a challenge. To address the requirements of future video codecs, this work proposes leveraging parallelism to increase the throughput for higher performance, which can be traded-off for lower power with voltage scaling. For existing standards such as H.264/AVC, the algorithms are fixed and parallelism can only be achieved through architecture optimizations. Since HEVC is still currently under development, there is an opportunity to jointly design the algorithm and architecture in order to achieve better results.

This work will focus on increasing the throughput of the video decoder. Specifically, it will address the throughput of the Context-based Adaptive Binary Arithmetic Coding (CABAC) engine, which is a key serial bottleneck that currently prevents a fully parallel video decoder from being achieved. This paper is organized as follows: Section II provides an overview of CABAC and highlights the features that make it difficult to parallelize. Section III describes several architecture and joint architecture-algorithm optimizations that can be used to reduce the critical path delay of the CABAC decoder. In Section IV, a new algorithm called Massively Parallel CABAC (MP-CABAC) is introduced, which leverages multiple forms of high level parallelism to increase the throughput while maintaining high coding efficiency. Optimizations to reduce area costs, memory size and external memory bandwidth are also discussed. Finally, Section V presents the measured results of the MP-CABAC decoder test chip.

II. Overview of CABAC

Context-based Adaptive Binary Arithmetic Coding (CABAC) is one of two entropy coding tools used in H.264/AVC; CABAC provides 9–14% improvement in coding efficiency compared to Huffman-based Context-based Adaptive Variable Length Coding (CAVLC) [2]. The high coding efficiency of CABAC can be attributed mainly to two factors. First, arithmetic coding performs better than Huffman coding, since it can compress closer to the entropy of a sequence by effectively mapping the syntax elements (e.g., motion vectors, coefficients, etc.) to codewords with non-integer number of bits; this is important when probabilities are greater than 0.5

Manuscript received April 13, 2011; revised June 19, 2011; accepted August 22, 2011. Date of current version December 23, 2011. This paper was approved by Guest Editor Tanay Karnik. This work was supported by Texas Instruments. Chip fabrication was provided by Texas Instruments. The work of V. Sze was supported by the Texas Instruments Graduate Women's Fellowship for Leadership in Microelectronics and NSERC.

V. Sze is with the Systems and Applications R&D Center, Texas Instruments, Dallas, TX 75432 USA (e-mail: sze@alum.mit.edu).

A. P. Chandrakasan is with the Microsystems Technology Laboratories, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: anantha@mtl.mit.edu).

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/JSSC.2011.2169310

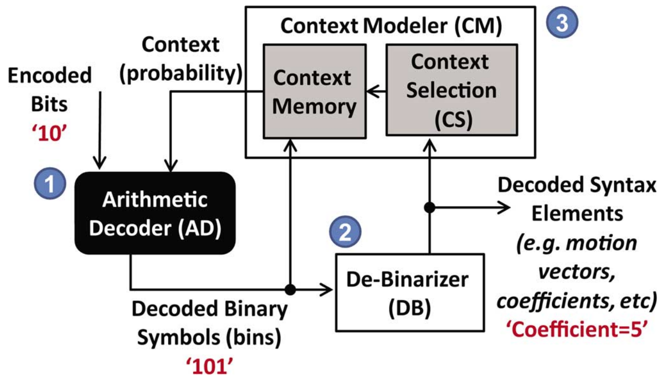


Fig. 1. The key blocks in the CABAC decoder.

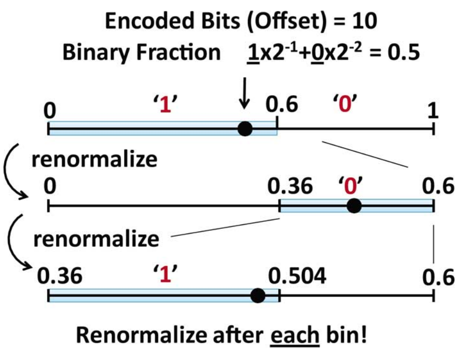

Encoded Bits (Offset) = 10
Binary Fraction 1x2⁻¹+0x2⁻² = 0.5
0 '1' 0.6 '0' 1
renormalize
0 0.36 '0' 0.6
renormalize
0.36 '1' 0.504 0.6
Renormalize after each bin!

Fig. 2. Arithmetic decoding example.

and the entropy is a fraction of a bit [3]. Second, CABAC is highly adaptive such that it can generate an accurate probability estimate, which results in better compression.

The main function of the CABAC decoder is to decode syntax elements from the encoded bits. The CABAC decoder is composed of three key blocks: arithmetic decoder (AD), de-binarizer (DB) and context modeler (CM). Fig. 1 shows the connections between these blocks. The AD decodes the binary symbol (bin) using the encoded bits and the probability of the bin that is being decoded. The probability of the bin is estimated using the CM. These bins are then mapped to syntax elements using the DB.

AD is based on recursive interval division as shown in Fig. 2. A range, with an initial value of 0 to 1, is divided into two subintervals based on the probability of the bin (e.g., $\Pr[\text{bin} = 0] = 0.4$ and $\Pr[\text{bin} = 1] = 0.6$). 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.

Next, the DB takes the decoded bins and maps them to a decoded syntax element. Various forms of mapping are used (e.g., unary, exp-golomb, fixed length) based on the type of syntax element being decoded.

Finally, the CM is used to generate an estimate of the bin's probability. An accurate probability estimate must be provided to the AD to achieve high coding efficiency. Accordingly, the

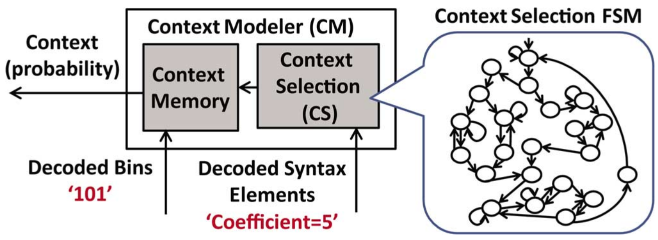


Fig. 3. Blocks in the context modeler.

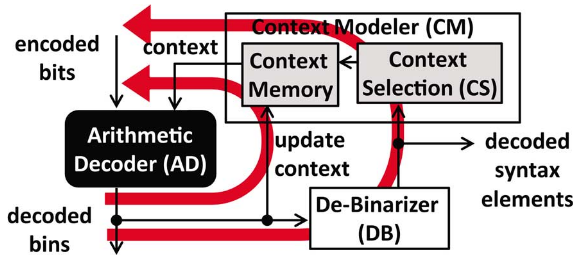


Fig. 4. Feedback loops in the CABAC decoder.

CM is highly adaptive and selects one of several hundred different contexts (probability models) depending on the type of syntax element, binIdx, luma/chroma, neighboring information, etc. This context selection (CS) is done using a large finite state machine (FSM) as shown in Fig. 3. A context switch can occur after each bin. The probability models are stored as 7-bit entries (6-bit for the probability state and 1-bit for the most probable symbol (MPS)) in a context memory and addressed using the context index computed by the CS FSM. Since the probabilities are non-stationary, the contexts are updated after each bin.

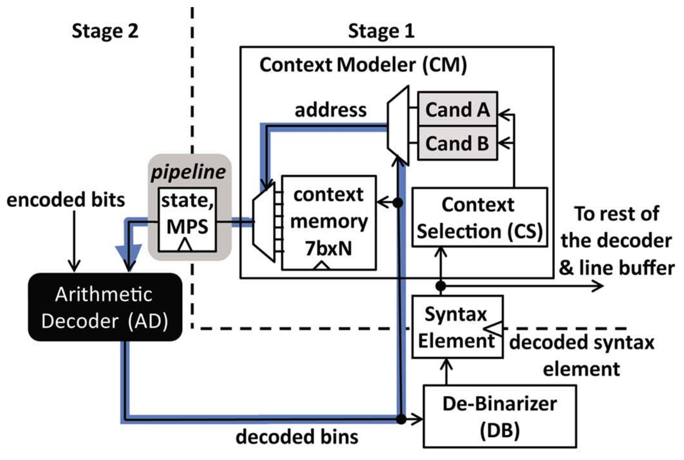
These data dependencies in CABAC result in tight feedback loops as shown in Fig. 4. Since the range and contexts are updated after every bin, the feedback loops are tied to bins; thus, the goal is to increase the overall bin-rate (bins per second) of the CABAC. In this work, two approaches are used to increase the bin-rate:

1) speed up the loop (increase cycles per second)
2) run multiple loops in parallel (increase bins per cycle)

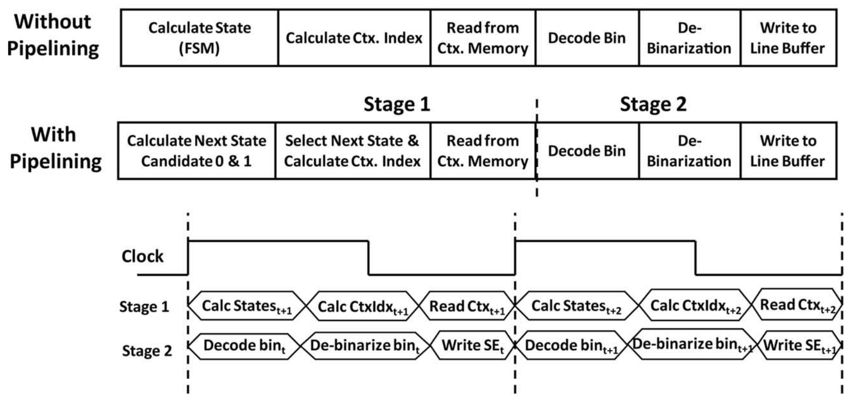
Due to these data dependencies, H.264/AVC CABAC implementations [4]–[7] need to use speculative computations to increase bins per cycle; however, the increase that can be achieved with this method is limited. Unlike the rest of the video decoder, which can use macroblock-line level (wavefront) parallelism, CABAC can only be parallelized across frames [8]; consequently, buffering is required between CABAC and the rest of the decoder, which increases external memory bandwidth [9]. By allowing the CABAC algorithm to also be optimized, further increase in bins per cycle can be achieved without additional cost to memory bandwidth.

III. Speed Up CABAC

Increasing the bin-rate of the CABAC can be achieved by reducing the delay of the feedback loops. This section discusses how pipelining the CABAC decoder as well as applying both architecture and joint architecture-algorithm optimizations to the arithmetic decoder (AD) can be used to reduce the critical path delay.



(a)



(b)

Fig. 5. Pipelining the CABAC to reduce the critical path delay. (a) Architecture of pipelined CABAC. (b) Timing diagram of the pipelined CABAC (Note: duration of operations not to scale.)

A. Pipelining CABAC

One of the key loops in the CABAC passes through all three blocks as shown in Fig. 5(a). Within a cycle, context selection is performed followed by arithmetic decoding and debinarization as shown in Fig. 5(b). The syntax element at the output of the debinarizer is registered and used by the rest of the decoder. To reduce the critical path delay of this loop, a pipeline register is inserted between the context memory of the context modeler and the arithmetic decoder. As a result, the context selection for the *next* bin is performed at the same time as the arithmetic decoding of the *current* bin. However, the context used for the next bin depends on the value of the current bin being decoded. To address this data dependency, while the arithmetic decoder (stage 2) is decoding the current bin, the two context candidates for the next bin are computed by the context selection FSM (stage 1). Once the current bin is decoded, it is used to select between the two context candidates for the next bin. The context index of the next bin is compared with the context index of the current bin. If they are the same, then the updated context state (i.e., probability) is used for the next bin. Otherwise, the context state is read from the memory.

Pipelining the CABAC in this manner reduces its critical path delay by approximately 40%. This architectural approach of pipelining and speculatively computing two context candidates to reduce the critical path can be used for the existing H.264/AVC CABAC. The context selection process involves two steps: 1) calculate the state transition of the FSM; 2) calculate the context index and read the candidate from the context memory. Speculative calculations are only done for the state transition (i.e., two possible transitions are computed, and one is selected based on decoded bin). The area cost for the additional context candidate computation logic is less than 3% of the total CABAC engine area and accounts for 4% the context selection power.

B. Optimization of Arithmetic Decoder

The data flow of the arithmetic decoder shown in Fig. 6. As mentioned earlier, arithmetic decoding involves recursively dividing the range into subintervals. The arithmetic decoder uses the probability of the bin to divide the range into two subintervals (the range of least probable symbol (LPS), rLPS, and the range of MPS, rMPS). To calculate the size of the subintervals, the H.264/AVC CABAC uses a look up table (LUT) called a modulo coder (M coder) rather than a true multiplier to reduce implementation complexity [10]. The offset is compared to the subintervals to make a decision about the decoded bin. It then

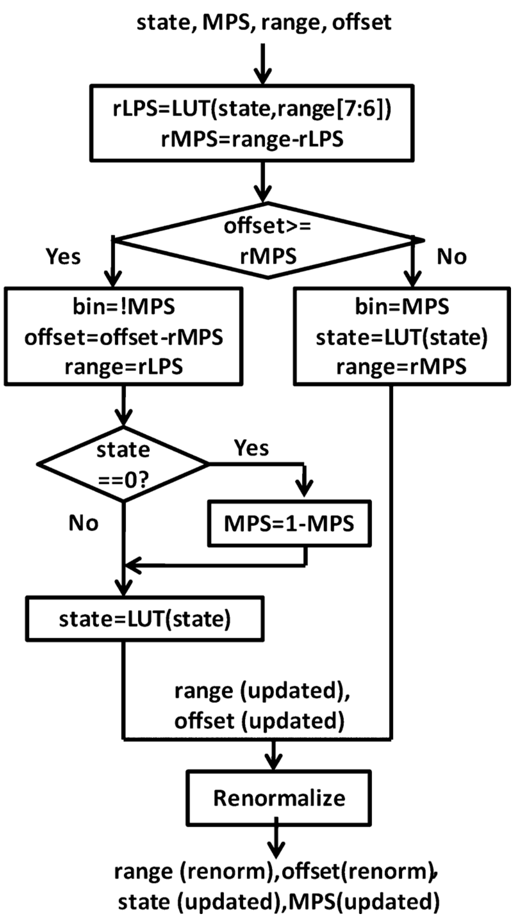


Fig. 6. Data flow in arithmetic decoder in H.264/AVC [12].

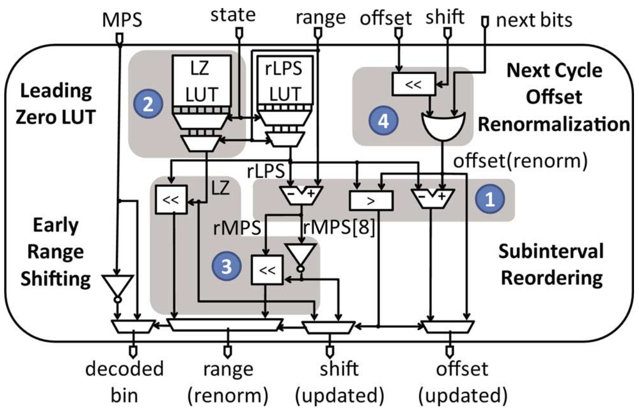


Fig. 7. Four optimizations of the arithmetic decoder to reduce critical path delay.

updates and renormalizes the range and sends the updated context back to the context memory.

Fig. 7 shows the architecture of the arithmetic decoder. The inputs to the arithmetic decoder include current context state (and MPS), range, offset, next bits, and shift (i.e., number of next bits). The outputs include updated context state (and MPS), updated range, updated offset, decoded bin and number of shifted bits due to renormalization.

Renormalization occurs in the critical path of the arithmetic decoder. Four optimizations are performed to speed up renormalization, increase concurrency and shorten the critical path delay of the arithmetic decoder as shown in Fig. 7: 1) subinterval reordering; 2) leading zero LUT; 3) early range shifting; and 4) next cycle offset renormalization. Subinterval reordering is a joint algorithm-architecture optimization, which requires changes to the algorithm and can be used in the yet to be finalized HEVC [11]. The other three optimizations 2), 3), and 4) are purely architectural, which means that they can also be applied to implementations of the existing H.264/AVC standard. The aggregate impact of these optimizations was a 22% reduction in the critical path delay of the AD. These optimization will be described in more detail in the next four sections.

1) Subinterval Reordering: In H.264/AVC, the rMPS is compared to the offset to determine whether the bin is MPS or LPS. The rMPS interval is computed by first obtaining rLPS from a 64×4 LUT (using bits [7:6] of the current 9-bit range and the 6-bit probability state from the context) and then subtracting it from the current range. The LUT contains constant values and is implemented with muxes. Depending on whether an LPS or MPS is decoded, the range is updated with their respective intervals. To summarize, the range division steps in the arithmetic decoder are as follows:

i) obtain rLPS from the 64×4 LUT;
ii) compute rMPS by subtracting rLPS from current range;
iii) compare rMPS with offset to make bin decoding decision;
iv) update range based on bin decision.

If the offset is compared to rLPS rather than rMPS, then the comparison and subtraction to compute rMPS can occur at the same time. Fig. 8 shows the difference between the range order of H.246/AVC CABAC and MP-CABAC. The two orderings of the intervals (i.e., which interval begins at zero, as illustrated in Fig. 8) are mathematically equivalent in arithmetic coding and thus changing the order has no impact on coding efficiency. This was also observed for the Q-coder in [13]. With this change, the updated offset is computed by subtracting rLPS from offset rather than rMPS. Since rLPS is available before rMPS, this subtraction can also be done in parallel with range-offset comparison. Changing the order of rLPS and rMPS requires the algorithm to be modified. Subinterval reordering was verified to have no coding penalty using the test model software of HEVC (HM-2.0) [14] under the common conditions set by the JCT-VC standards body [15]. This optimization accounts for half of the overall 22% critical path reduction.

2) Leading Zero LUT: After the range is updated based on the bin decision, renormalization may be necessary for the range and offset due to the use of finite bit precision. Renormalization involves determining the number of leading zeros (LZ) in the updated range and shifting the range accordingly (Fig. 9). LZ can be determined through the use of muxes in the form of a priority encoder. However, using a serial search for the first non-zero can increase the critical path delay.

If an LPS is decoded, the updated range is rLPS and renormalization must occur. Recall that rLPS is stored in a 64×4 LUT implemented with muxes, indexed by the probability state and bits [7:6] of the original range. Since every rLPS can be mapped to a given LZ, an LZ LUT can be generated that is also indexed

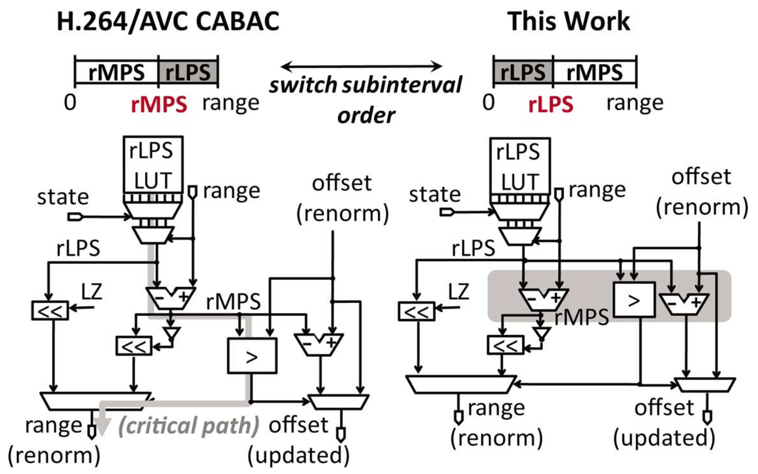


Fig. 8. Proposed joint algorithm-architecture optimization called subinterval reordering, which reduces critical path delay with no coding efficiency loss.

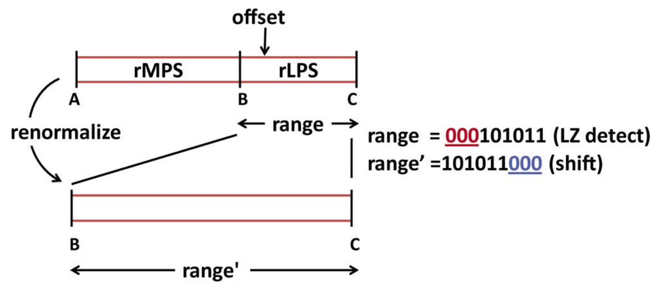


Fig. 9. Renormalization in arithmetic decoder.

by the probability state and original range. This enables LZ to be determined in parallel with rLPS and reduces the critical path by avoiding the serial priority encoder. The LZ LUT has the same number of entries (64×4) as the rLPS LUT, but each entry is only 3-bits (for a max shift of 7) compared with 9-bits for rLPS LUT; thus the LZ LUT is approximately 1/3 of the rLPS LUT size. The LZ LUT accounts for 3% of the total arithmetic decoder area and 2% of the arithmetic decoder power.

If an MPS is decoded, LZ can only be determined after the rMPS subtraction. However, LZ can be quickly determined from the most significant bit (MSB) of the updated range rMPS and thus has little impact on the critical path.

3) Early Range Shifting: After a decision is made on the decoded bin, and LZ is determined, the range is shifted to the left based on LZ. The shifting can be implemented using shift registers; however, this approach of moving one bit per cycle results in up to 7 clock cycles per renormalization (for the minimum range value of 2). Alternatively, the shifting can be done through the use of combinational logic (e.g., 9:1 mux) which can be done in a single cycle, but may increase the critical path.

To mitigate this, the shifting can be done *before* the decoded bin is resolved. Specifically, rLPS is shifted in parallel with the range-offset comparison and rMPS subtraction described earlier. rMPS is shifted by a maximum of one, which can be done quickly after the rMPS subtraction. Once the decoded bin is resolved, the range can be immediately updated with the renormalized rLPS or rMPS.

4) Next Cycle Offset Renormalization: Offset renormalization involves a left shift by the same amount based on the LZ of the range, and new bits are appended to the right. The offset is not used until after rLPS is determined. Therefore, rather than performing this shift in the current cycle after the bin is resolved, the shifting operation can be moved to the *next* cycle where it is done in parallel with the rLPS look up.

IV. Massively Parallel CABAC

The bin-rate can also be increased by processing multiple bins per cycle. However, due to feedback loops, processing multiple bins per cycle requires speculative computations. Instead of trying to process more bins per cycle within an arithmetic decoder, this work proposes replicating the loop and running many arithmetic decoders in parallel. It is also important that the high coding efficiency of CABAC be maintained. For instance, in H.264/AVC a frame can be divided into several regular slices which can be processed in parallel. These H.264/AVC slices contain macroblocks (i.e., 16×16 blocks of pixels) that are coded completely independently from other slices making them suitable for parallel processing. However, since the H.264/AVC slices are independent, no redundancy between the slices can be removed which leads to significant coding loss.

Massively Parallel CABAC (MP-CABAC), previously developed by the authors [16], is currently under consideration for HEVC, and has been adopted into the standard body's JM-KTA working software [17]. It enables parallel processing, while maintaining the high coding efficiency of CABAC. MP-CABAC has a more efficient coding efficiency to throughput trade-off than H.264/AVC slices as shown in Fig. 10(a). For a $10\times$ increase in throughput, MP-CABAC has a $4\times$ lower coding penalty than H.264/AVC slices. Note that MP-CABAC also has an improved trade-off between throughput and area cost compared with H.264/AVC slices, due to better workload balancing and reduced hardware replication (Fig. 10(b)).

MP-CABAC uses a combination of two forms of parallelism: Syntax Element Partitions (SEP) and Interleaved Entropy Slices (IES). SEP enables different syntax elements (e.g., motion vectors, coefficients, etc.) to be processed in parallel with low area

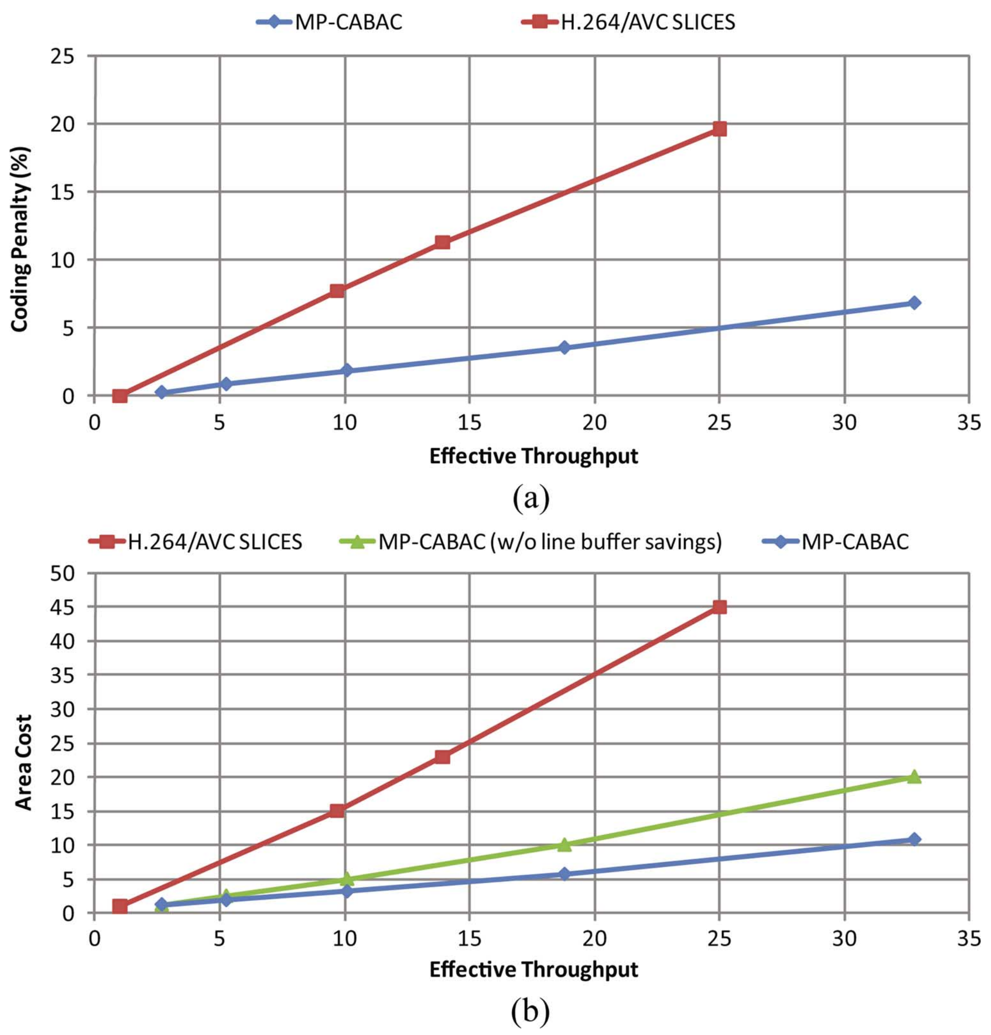


Fig. 10. Trade-off comparison between MP-CABAC and H.264/AVC slices measured under common conditions specified by the standardization body [20]. Coding efficiency and throughput are averaged across prediction structures, sequences and quantization. To account for any workload imbalance, the slice with the largest number of bins per frame was used to compute the throughput. (a) Coding efficiency versus throughput trade-off. (b) Area versus throughput trade-off.

cost [18]. IES enables several slices to be processed in parallel, allowing the entire decoder to achieve wavefront parallel processing without increasing external memory bandwidth [19]. SEP and IES will be described in more detail the next two sections.

A. Syntax Element Partitions (SEP)

Syntax Element Partitions (SEP) is a method distributing the bins across parallel arithmetic decoders based on the syntax element [18]. This approach has both coding efficiency and area cost benefits.

One of the features that gives CABAC its high coding efficiency is that the contexts are adaptive. While encoding/decoding, the contexts undergo training to achieve an accurate estimate of the bin probabilities. A better estimate of the probabilities results in better coding efficiency. A drawback of breaking up a frame into slices is that there are fewer macroblocks, and consequently fewer syntax elements, per slice. Since the entropy engine is reset every slice, the context undergoes less training and can results in a poorer estimate of the probabilities.

To avoid reducing the training, rather than processing slices in parallel, syntax elements are processed in parallel. In other words, rather than grouping bins by macroblock and placing them in different slices, bins are grouped based on syntax element and placed in different partitions which are then processed in parallel (Fig. 11).[1] As a result, each partition contains all the bins of a given syntax element, and the context can then undergo the maximum amount of training (i.e., across all occurrences of the element in the frame) to achieve the best possible probability estimate and eliminate the coding efficiency penalty from reduced training. Table I shows the five different partitions of syntax elements. The syntax elements were assigned to partitions based on the bin distribution in order to achieve a balanced workload. It should be noted that as the number of partitions increases, it becomes more difficult to ensure a balanced workload across partitions, which limits the throughput that can be achieved with this form of parallelism. Techniques such as those proposed in [18], where partitions are adaptively recombined, can be used to improve workload balance. A start

[1]SEP are color coded in Figs. 11, 12, 13, 15, 20 and 21.

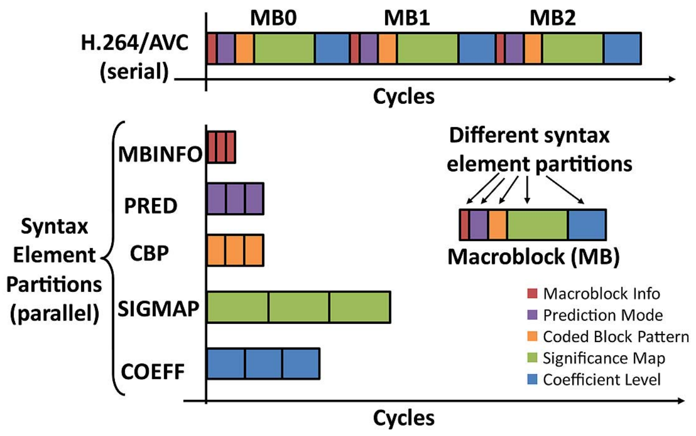


Fig. 11. Cycle reduction is achieved by processing SEP in parallel.

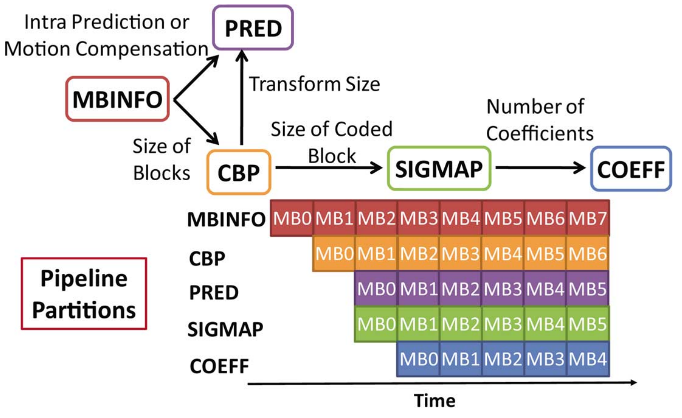

Intra Prediction or Motion Compensation
PRED
MBINFO
Transform Size
Size of Blocks
CBP
Size of Coded Block
SIGMAP
Number of Coefficients
COEFF
Pipeline Partitions
MBINFO MB0 MB1 MB2 MB3 MB4 MB5 MB6 MB7
CBP MB0 MB1 MB2 MB3 MB4 MB5 MB6
PRED MB0 MB1 MB2 MB3 MB4 MB5
SIGMAP MB0 MB1 MB2 MB3 MB4 MB5
COEFF MB0 MB1 MB2 MB3 MB4
Time

Fig. 12. Dependencies between SEP requires that partitions be pipelined and synchronized.

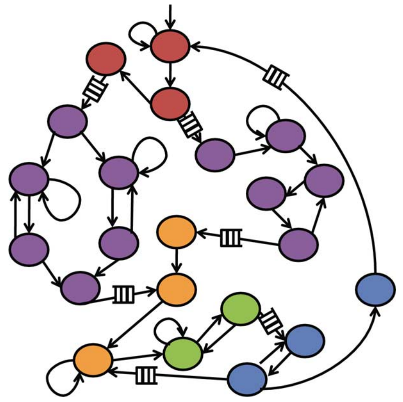


Fig. 13. Context Selection FSM is divided into smaller FSM for each SEP and FIFOs are used for synchronization.

code prefix for demarcation is required at the beginning of each partition.

There are dependencies between syntax elements in different partitions as shown in Fig. 12. For instance, the type of prediction (spatial or temporal), which is signaled by mb_type in the MBINFO partition, needs to be known in order to determine whether motion vectors or intra prediction modes should be decoded in the PRED partition. To address this, a pipeline architecture is used such that different macroblocks for different partitions are processed at the same time. For instance, the MBINFO partition for a given macroblock (MB0) must be decoded before the PRED partition for the same macroblock (MB0). However, the MBINFO partition of MB2 can be decoded in parallel with the PRED partition of MB0 as shown in Fig. 12. Thus, the processing of each partition must be synchronized. Synchronization can be done using data driven first-in-first-out queues (FIFOs) between engines, similar to the ones used in [21] and [22] between processing units.

TABLE I
SYNTAX ELEMENT PARTITIONS

Partition	Syntax Element
MBINFO	mb_skip_flag, mb_type, sub_mb_type, end_of_slice_flag
PRED	prev_intra4x4_pred_mode_flag, rem_intra4x4_pred_mode, prev_intra8x8_pred_mode_flag, rem_intra8x8_pred_mode, intra_chroma_pred_mode, ref_idx_l0, ref_idx_l1, mvd_l0, mvd_l1
CBP	transform_size_8x8_flag, mb_qp_delta, coded_block_pattern, coded_block_flag
SIGMAP	significant_coeff_flag, last_significant_coeff_flag
COEFF	coeff_abs_level_minus1, coeff_sign_flag

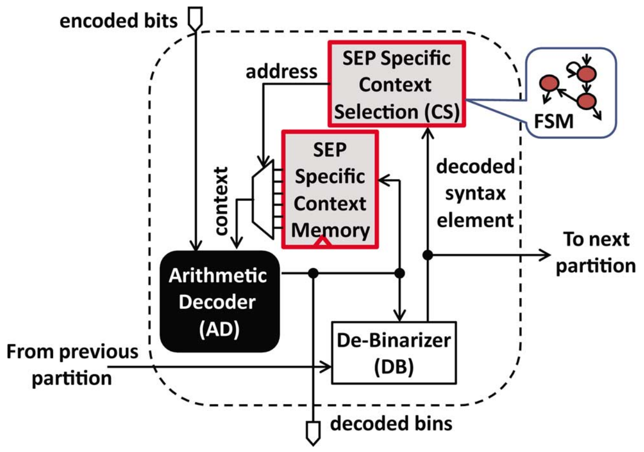


Fig. 14. Partition engine (PE) composed of small SEP specific context selection FSM and context memory.

The hardware required to decode five SEP in parallel can be implemented with relatively low area cost. The FSM of the context modeler (CM) and de-binarizer (DB) is divided into smaller FSMs for each SEP. These small FSM remain connected by FIFOs to synchronize the different partitions as shown in Fig. 13. The register-based context memory is divided into smaller memories for each SEP so that the number of storage elements remain the same. While the number of memory ports increases, the address decoder for each port is smaller than the one used for the single large memory and thus the area of the context memory is only increased by around 26%. Accounting

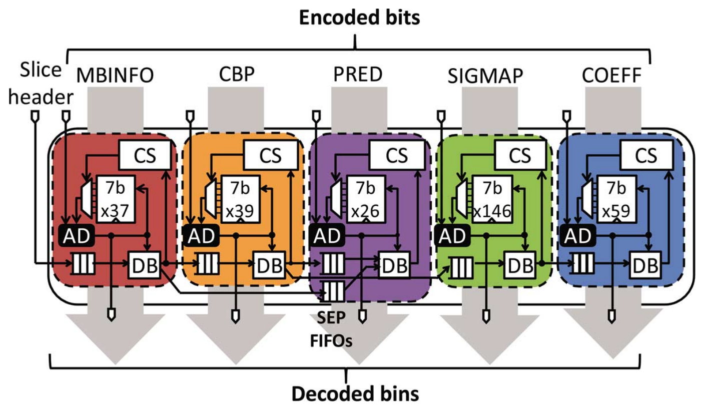


Fig. 15. Slice engine composed of five partition engines that operate concurrently on five different SEP.

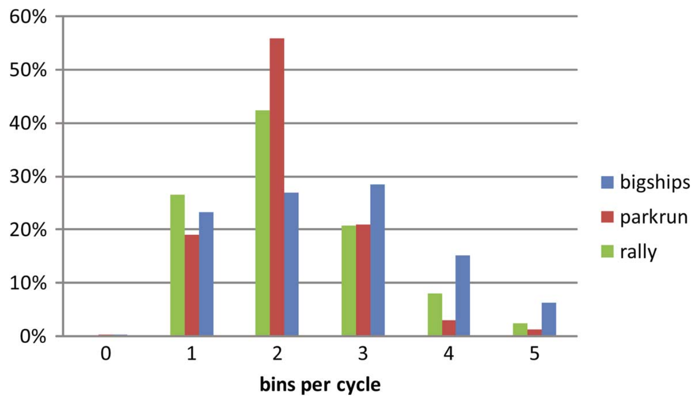


Fig. 16. Bins per cycle distributions for different sequences using the slice engine architecture.

for the additional context memory ports, the FIFOs and the replicated AD, $5\times$ SEP parallelism can be achieved by increasing the overall area by only 70%.

Five partition engines (PEs) are formed from the small FSM, context memory and AD and operate in parallel to process the five different SEP (Fig. 14). Speculative computations for state transition in the FSM, discussed in Section III-A, can also be used in partition engines (note: logic is omitted in Fig. 14 for simplicity). In fact, the smaller FSM for each partition engine may help to reduce the power overhead of the speculative computations. These five partition engines are combined to form a slice engine shown in Fig. 15. During the stall cycles, the partition engine clock is disabled with hierarchical clock gating to reduce power. Using this slice engine architecture, up to five bins can be decoded in parallel with an average throughput increase of $2.4\times$. The bins per cycle distribution of various sequences for the slice engine is shown in Fig. 16.

B. *Interleaved Entropy Slices (IES)*

To achieve additional throughput improvement, the previous approach can be combined with Interleaved Entropy Slices (IES). For the purpose of highlighting the differences between H.264/AVC slices and IES, the video decoder can be broken into two parts: the entropy decoding portion (which will be referred to as entropy decoder), and the rest for the decoder (which will be referred to as pixel decoder). Furthermore, the definition of *dependencies* in the subsequent discussion refers to the top and left neighboring macroblocks.

As mentioned earlier, in H.264/AVC, a frame can be broken into regular slices. In most cases,[2] the slices are independent of each other, meaning each slice can be fully decoded (i.e., reconstruct all pixels) without any information from the other slices. This allows regular slices to be fully decoded in parallel. Accordingly, the entropy decoder and pixel decoder can run in parallel. One key drawback of having slices that are entirely independent is that redundant information cannot be removed across slices, which results in coding loss.

Entropy slices, introduced in [23], enable independent entropy decoding, where all the syntax elements can be decoded without information from other entropy slices. However, to achieve better coding efficiency than fully independent slices (i.e., H.264/AVC slices), there remains dependencies between the entropy slices when using the syntax elements to decode pixels (e.g., for spatial and motion vector prediction). In other words, the entropy decoder can run in parallel; however, it does not enable parallelism in the pixel decoder. In order for both the entropy decoder and pixel decoder to operate in parallel, frame level buffering is required which increases external memory bandwidth [9].

Interleaved entropy slices (IES) allow dependencies across slices for both syntax element and pixel decoding which improves coding efficiency [19]. To enable parallel processing of slices that have dependencies, IES divides a frame into slices in a different manner than entropy slices and H.264/AVC slices. A typical spatial location of the macroblock allocated to H.264/AVC slices and entropy slices is shown in Fig. 17(a). For IES, the macroblocks are allocated as shown in Fig. 17(b). Different rows of macroblocks are assigned to each slice. As long as slice 0 is one or two macroblocks ahead of slice 1, both slices can be decoded in parallel; similarly, slice 1 must be ahead of slice 2 and slice 2 must be ahead of slice 3. This form of processing is often referred to wavefront processing. With interleaved entropy slices, both the entropy decoder and pixel decoder can process different slices in parallel. Consequently, no buffering is required to store the decoded syntax elements, which reduces memory costs. In other words, IES allows the entire decoder to achieve wavefront parallel processing without increasing external memory bandwidth. This can have benefits in terms of reducing system power and possibly improving performance (by avoiding read conflicts in shared memory). Using IES to parallelize the entire decoder path can improve the overall throughput and reduce the power of the entire video decoder [19].

IES are processed in parallel by several slice engines as shown in Fig. 19. IES FIFOs are used between slice engines to synchronize IES required due to top block dependencies. The properties of the neighboring blocks are used for context selection and are stored in the IES FIFOs and line buffer. Section IV-D will discuss a joint algorithm-architecture optimization in the context selection logic that reduces the line buffer size.

[2]An exception is when deblocking is enabled across regular slices.

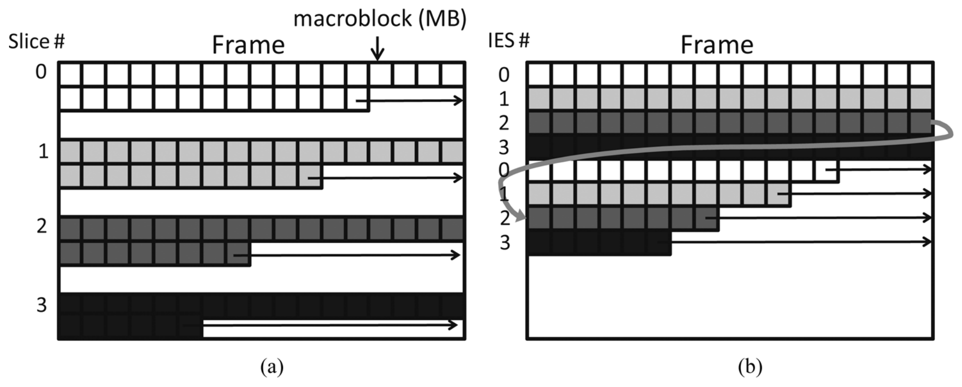


Fig. 17. Macroblock allocation to different slices. (a) H.264/AVC slices. (b) Interleaved entropy slices.

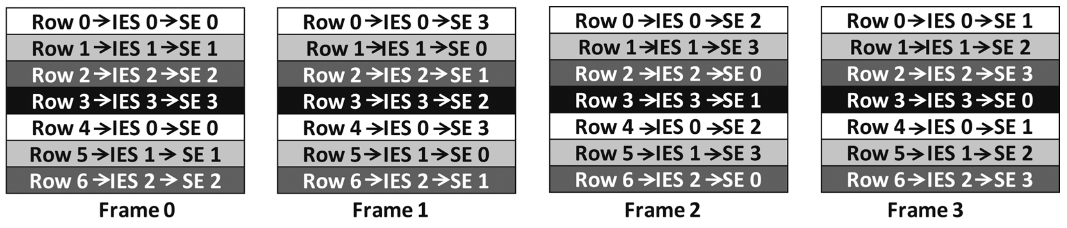


Fig. 18. Example of row balancing when the number of rows in a frame (e.g., 7) is not a multiple of the number of slices (e.g., 4).

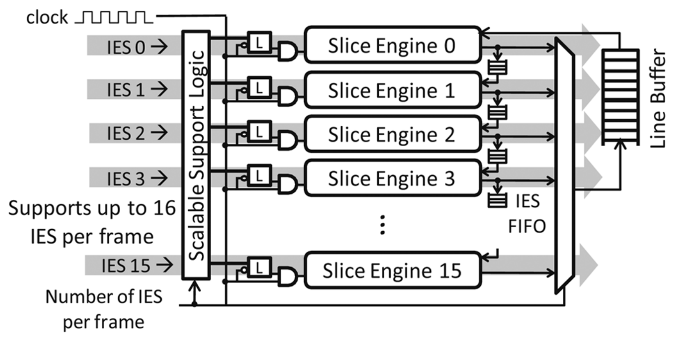


Fig. 19. Architecture for IES.

The number of accesses to the large line buffer is also reduced for IES. In Fig. 17(b), the line buffer (which stores an entire macroblock row) is only read by slice 0 (and written by slice 3). Since slice 0 is only several macroblocks ahead of slice 1, slice 1 only needs to access a small cache (IES FIFO), which stores only a few macroblocks, for its last line (top) data. Thus out of N slices, N-1 will access small FIFOs for the last line data, and only one will access the large line buffer. If the line buffer is stored on-chip, interleaved entropy slices reduces area cost since it does not need to be replicated for every slice as with H.264/AVC and entropy slices. Alternatively, if the line buffer is stored off-chip, the off-chip memory bandwidth for last line access is reduced to 1/N of the H.264/AVC line buffer bandwidth.

Finally, in interleaved entropy slices the number of bins per slice (i.e., workload) tends to be more equally balanced; consequently, a higher throughput can be achieved for the same amount of parallelism. Workload imbalance can also occur when the number of rows in a frame is not a multiple of the number of slices (i.e., the number of macroblocks per slice is not equal). To address this, row balancing is used, where the IES assigned to each slice engine rotates for each frame resulting in up to a 17% increase in throughput. Fig. 18 shows an example of how row balancing can used to balance a frame with 7 rows across 4 slice engines (SE). Better workload balancing and avoiding line buffer replication improves the trade-off between throughput and area as shown in Fig. 10(b).

To enable scalability, the number of slice engines is configurable; a multiplexer connects the output of the last enabled slice engine to the line buffer. To reduce power, the clocks to the disabled slice engines are turned off using hierarchal clock gating. Over 9× increase in throughput is achieved with 16 IES per frame using the architecture in Fig. 19.

To summarize, the benefits of interleaved entropy slices include the following:

- improved coding efficiency over H.264/AVC slices;
- simple synchronization with IES FIFOs;
- reduction in memory bandwidth;
- improved workload balance;
- fully parallel video decoder.

In subsequent HEVC proposals [24], [25], IES has been extended to include context initialization dependencies across slices to improve coding efficiency; a multithreaded implementation of this combined approach was demonstrated and the extended version of IES has been adopted into the working draft 4 of the HEVC standard [26].

C. *Data Structure*

Fig. 20 shows the structure of the encoded data which describes the frames in the video sequence. For the MP-CABAC, each frame is composed of several IES and each IES is composed of five SEP. A 32-bit startcode is inserted at the beginning of each partition to enable the parser to access any partition within the bitstream. The partitions can then be distributed across several engines to be processed in parallel. The slice

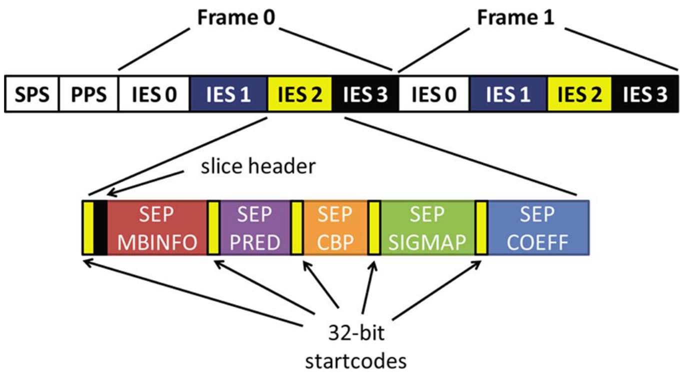


Fig. 20. MP-CABAC data structure. In this example, there are four IES per frame and five SEP per IES.

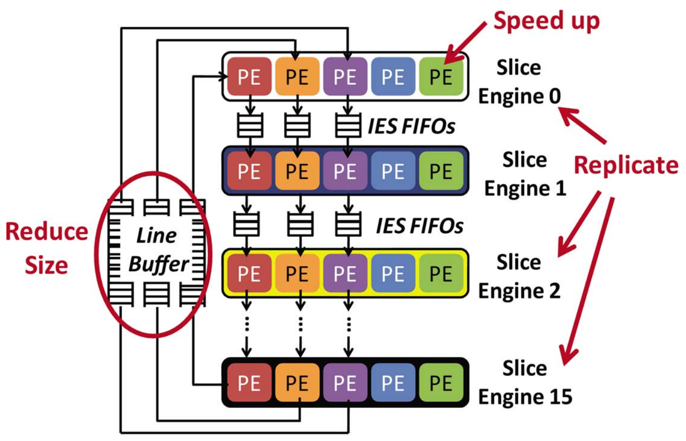


Fig. 21. Optimizations performed on MP-CABAC architecture. (PE = partition engine).

header information, such as slice type (I, P, B), slice quantization, etc., is inserted at the beginning of the MBINFO partition. The MP-CABAC test chip presented in this paper supports up to 16 IES per frame with 80 arithmetic decoders running in parallel.

D. Line Buffer Reduction

Fig. 21 shows the top level MP-CABAC architecture used to decode the data structure in Fig. 20 and highlights the optimizations discussed in this paper. Section III discussed optimizations that were performed to speed up the AD, while Sections IV-A and IV-B described how AD can be replicated to run in parallel while still maintaining high coding efficiency. This section will discuss how to reduce the size of the line buffer.

To make use of the spatial correlation of neighboring data, context selection can depend on the values of the top and left blocks as shown in Fig. 22. Consequently, a line buffer is required in the CABAC engine to store information pertaining to the previously decoded row. The depth of this buffer depends on the width of the frame being decoded which can be quite large for high resolution (e.g., QFHD) sequences. The bit-width of the buffer depends on the type of information that needs to be stored per block or macroblock in the previous row. This section discusses a joint algorithm-architecture optimization that reduces the bit-width of this data to reduce the overall line buffer size of the CABAC.

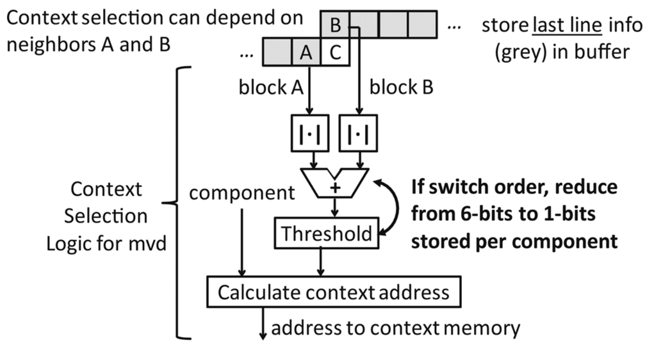

Context selection can depend on neighbors A and B
B
A
C
store last line info (grey) in buffer
block A
block B
|·|
|·|
+
Context Selection Logic for mvd
component
Threshold
If switch order, reduce from 6-bits to 1-bits stored per component
Calculate context address
address to context memory

Fig. 22. Modified context selection for *mvd*.

Specifically, this work proposes modifying the context selection for *motion vector difference* (*mvd*). *mvd* is used to reduce the number of bits required to represent motion information. Rather than transmitting the motion vector, the motion vector is predicted from its neighboring 4×4 blocks and only the difference between motion vector prediction (*mvp*) and motion vector (*mv*), referred to as *mvd*, is transmitted.

$$mvd = mv - mvp$$

A separate *mvd* is transmitted for the vertical and horizontal components. The context selection of *mvd* depends on neighbors A and B as shown in Fig. 22. For position C, context selection is dependent on A and B (4×4 blocks for *mvd*); a line buffer required to store the previous row of decoded data.

In H.264/AVC, neighboring information is incorporated into the context selection by adding a context index increment (between 0 to 2 for *mvd*) to the calculation of the context index. The *mvd* context index increment, χ_{mvd}, is computed in two steps [2]:

Step 1: Sum the absolute value of neighboring *mvd*s

$$\mathrm{e(A,B,cmp)} = |mvd\,(\mathrm{A,cmp})| + |mvd(\mathrm{B,cmp})|$$

where A and B represent the left and top neighbor and cmp indicates whether it is a vertical or horizontal component.

Step 2: Compare e(A, B, cmp) to thresholds of 3 and 32

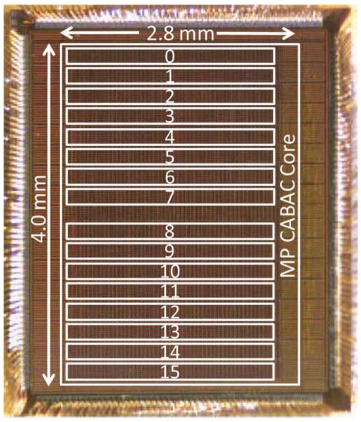
$$\chi_{mvd}\,(\mathrm{cmp}) = \begin{cases} 0, & \text{if } e(\mathrm{A,B,cmp}) < 3 \\ 1, & \text{if } 3 \le \mathrm{e(A,B,cmp)} \le 32 \\ 2, & \text{if } e(\mathrm{A,B,cmp}) > 32. \end{cases}$$

With the upper threshold set to 32, a minimum of 6 bits of the *mvd* has to be stored per component per 4×4 block in the line buffer. Certain blocks may be bi-predicted which means up to two motion vectors are required per block. For QFHD, there are $(4096/4) = 1024$ $4{\times}4$ blocks per row, which implies $6 \times 2 \times 2 \times 1024 = 24{,}576$ bits are required for *mvd* storage.

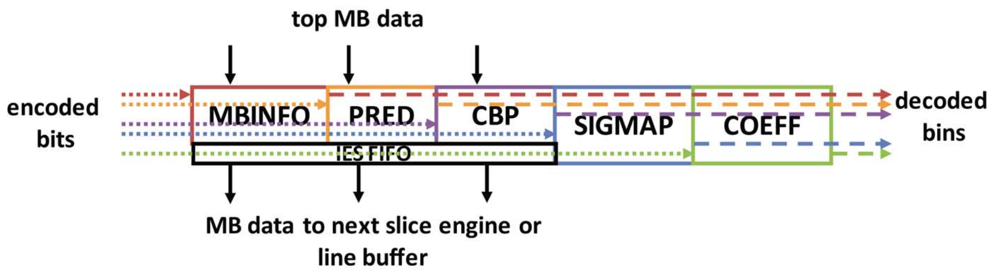
To reduce the memory size, rather than summing the components and then comparing to a threshold, this work proposes separately comparing each component to a threshold and summing their results. In other words:

Step 1: Compare the components of *mvd* to a threshold

$$\mathrm{thresh}_A\,(\mathrm{cmp}) = |mvd\,(\mathrm{A,cmp})| > 16$$
$$\mathrm{thresh}_B\,(cmp) = |mvd\,(\mathrm{B,cmp})| > 16.$$



(a)



(b)

Fig. 23. Die micrograph and floorplan of test chip. (a) Die micrograph. 16 slice engines are highlighted. (b) Floorplan of the slice engine.

Step 2: Sum the results thresh_A and thresh_B from Step 1

$$\chi_{mvd}\,(cmp) = \mathrm{thresh}_A\,(\mathrm{cmp}) + \mathrm{thresh}_B\,(\mathrm{cmp}).$$

A single threshold of 16 is used. Consequently, only a single bit is required to be stored per component per 4×4 block; the size of the line buffer for *mvd* is reduced to $1 \times 2 \times 2 \times 1024 = 4,096$ bits. In H.264/AVC, the overall line buffer size of the CABAC required for all syntax elements is 30,720 bits. The modified *mvd* context selection reduces the memory size by 67%, from 30,720 bits to 10,240 bits. This optimization has negligible impact ($\leq 0.02\%$) on coding efficiency [11].

V. Results

The MP-CABAC test chip shown in Fig. 23(a) was implemented in 65-nm CMOS [27]. The floorplan of the slice engine is shown in Fig. 23(b). Table II shows a summary of the chip features. The MP-CABAC test chip contains 80 arithmetic decoders running in parallel. A round robin interface is used to move data between these parallel engines and the I/O of the chip [28].

Table III compares the MP-CABAC test chip against existing state-of-the-art H.264/AVC CABAC implementations. MP-CABAC achieves an average of 24.11 bins/cycle across several HD video sequences, which is 10.6× higher than existing H.264/AVC CABAC implementations [4]–[7]. Note that the throughput of these H.264/AVC CABAC implementations are limited by the fixed H.264/AVC algorithm. The MP-CABAC approach can be combined with techniques used in [4]–[7] for additional throughput increase of 1.32 to 2.27× on top of the 24.11 bins/cycle.

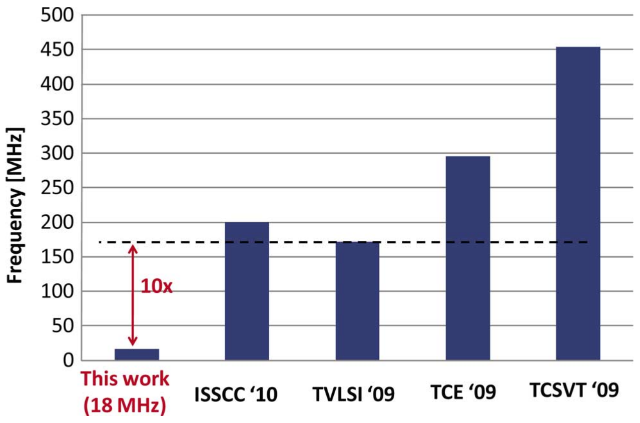


Fig. 24. Comparison of the minimum frequency require to decode 300 Mb/s sequences. Lower frequency implies additional voltage scaling can be used for low power applications.

At 1.0 V, it has a performance of 3026 Mbins/s for a bit-rate of 2.3 Gbps, enough for real-time QFHD at 186 fps, or equivalently 7.8 streams of QFHD at 24 fps. For low power applications, the MP-CABAC test chip decodes the max H.264/AVC bit-rate (300 Mb/s) with a 18 MHz clock at 0.7 V, consuming only 12.3 pJ/bin (Fig. 24).

Fig. 25 shows the trade-off between measured power, performance (bin-rate) and coding efficiency across a wide operating range. Scaling the number of IES per frame from 1 to 16 increases the performance range by an order or magnitude, and reduces the minimum energy per bin by 3× to 10.5 pJ/bin with less than a 5% coding penalty. Note that the metric used to measure the energy efficiency (performance/watt) of the CABAC

TABLE II
SUMMARY OF CHIP IMPLEMENTATION

Video Coding Standard	Massively Parallel CABAC proposed for HEVC
Technology	65-nm CMOS
Transistor Count	9.74 M (includes all memory)
Core Area	2.8 mm $\times$ 4.0 mm
Max Resolution	4096$\times$2160
Supply Voltage	0.6 to 1.0 V (core) 1.8 V (I/O)
Operating Frequency	5.5 to 125.5 MHz (core)
Bin-rate	14 to 3026 Mbins/s
Core Power	0.6 to 77 mW
Energy Metric	10.5 pJ/bin @ 0.6 V (Min Energy Point) 12.3 pJ/bin @ 0.7 V (Max H.264/AVC bit-rate)

TABLE III
COMPARISON OF MP-CABAC WITH STATE-OF-THE-ART H.264/AVC CABAC IMPLEMENTATIONS. NOTE: APPROACHES IN THIS WORK ARE COMPLEMENTARY TO BIN LEVEL APPROACHES IN THESE OTHER IMPLEMENTATIONS AND CAN BE COMBINED FOR HIGHER OVERALL PERFORMANCE

	This Work	**ISSCC'10 [4]**	**TCE'09 [5]**	**TVLSI'09 [6]**	**TCVST'09 [7]**
Standard	proposed for HEVC	H.264/AVC	H.264/AVC	H.264/AVC	H.264/AVC
Peak bins/cycle	80	2	2	16	2
Average bins/cycle	24.11	1.95	1.32	2.27	0.86
Coding Penalty[a]	$<5\%$	none	none	none	none
Process (nm)	65	90	130	180	180
Voltage	1.0	1.0	1.2	1.8	1.8
Results	Measured	Measured	Simulated	Simulated	Simulated
Maximum Frequency [MHz]	125.5	210	238	45	105
Performance [Mbins/s]	3026	410	314	102	90

[a]compared with H.264/AVC single slice per frame

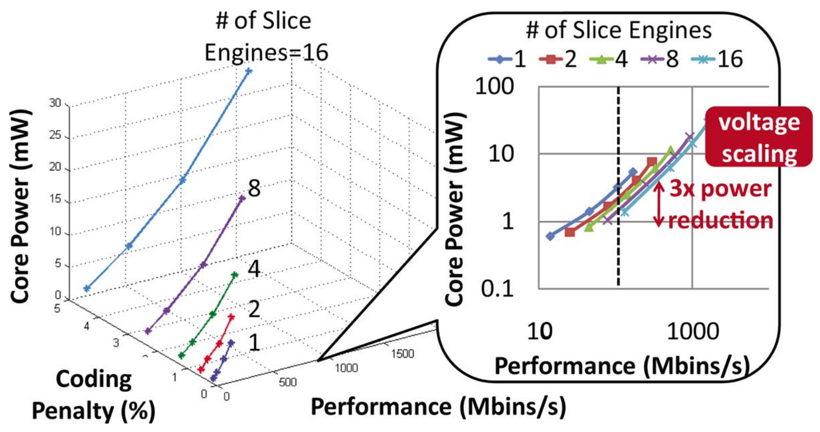


Fig. 25. Trade-off between coding efficiency, power and performance (bin-rate).

is in terms of bins/s/mW (i.e., pJ/bin) rather than pixels/s/mW, which is traditionally used for full video decoders. pJ/bin is used since the performance of CABAC is dictated by the bin-rate (Mbins/s), which can vary widely for a given frame rate and resolution (pixels/s).

Figs. 26 and 27 shows the power and area breakdown of the core, slice engine and across partition engines. All memories (e.g., context memory, line buffer, SEP FIFO, and IES FIFO) were implemented with registers. Additional increase in throughput can be achieved by increasing the depth of the SEP

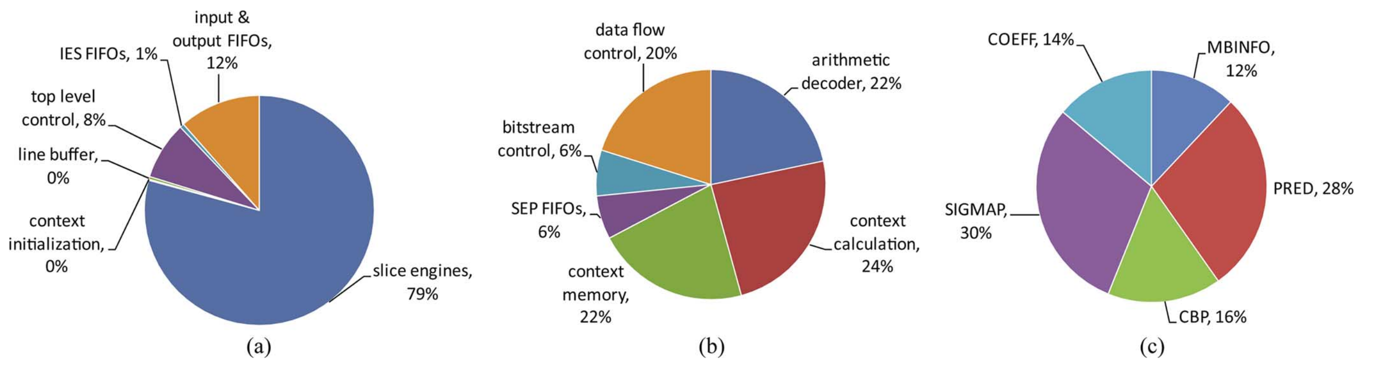


Fig. 26. Simulated (post-layout) power breakdown of MP-CABAC. (a) Breakdown within MP-CABAC core. (b) Breakdown within a slice engine. (c) Breakdown across partitions.

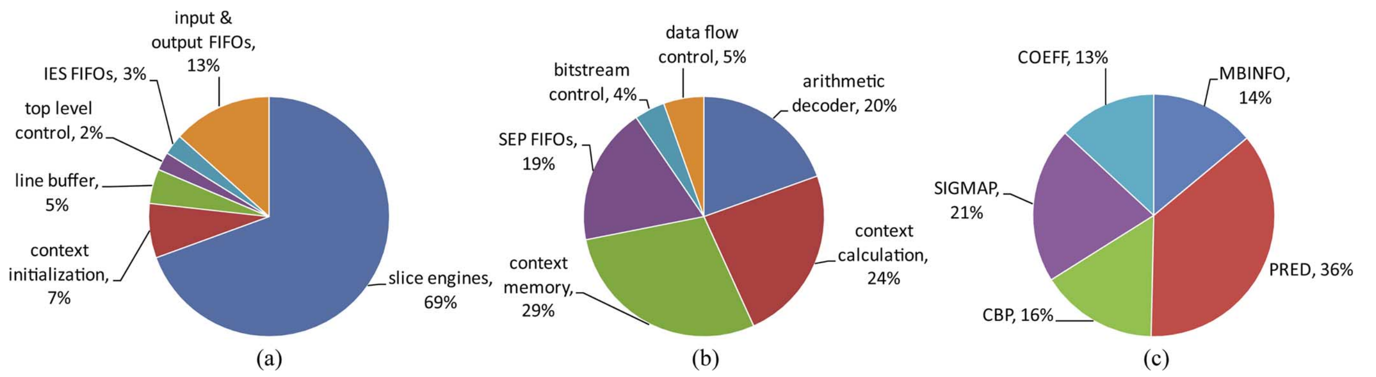


Fig. 27. Synthesis area breakdown of MP-CABAC. (a) Breakdown within MP-CABAC core. (b) Breakdown within a slice engine. (c) Breakdown across partitions.

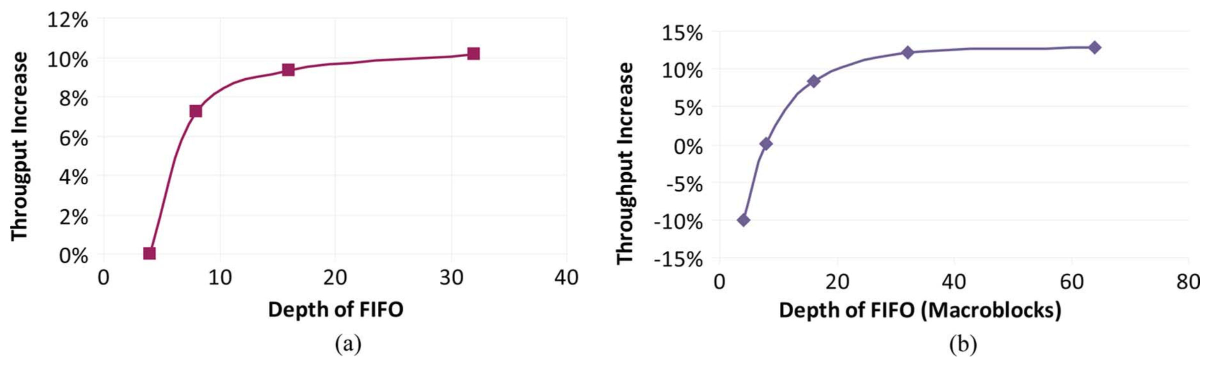


Fig. 28. Trade-off between FIFO depth and throughput. (a) SEP FIFO. (b) IES FIFO.

FIFO and IES FIFOs as shown in Fig. 28. However, this comes at the cost of increased area. Input and output FIFOs in Figs. 26 and 27 refer to the bitstream and decoded bin buffers. The context initialization was designed to perform initialization of all probability models within a cycle; the context initialization area in Fig. 27(a) can be reduced by make the initialization process serial.

This work presents several methods of increasing throughput (performance) with minimal overhead to coding efficiency. The throughput can then be traded-off for power savings via voltage scaling. As previously discussed, a measured $3\times$ power reduction was achieved by using 16 IES which increased throughput by over $9\times$. From the voltage-delay relationship, the power impact from SEP, pipelining CABAC and optimizing AD can be estimated to be $1.6\times$, $1.3\times$, and $1.1\times$, based solely on voltage scaling from nominal supply voltage. Note that the amount of power savings for a given increase in throughput depends on the operating point on the voltage-delay curve. Thus, while the throughput benefits of each innovation are mostly cumulative, the power savings are not. Since throughput is traded-off for power savings, both throughput and power benefits cannot be achieved simultaneously.

VI. Summary

Both power and performance demands will continue to rise for future video codecs. It will be increasingly difficult to meet these demands with architecture optimizations alone. The MP-CABAC test chip presented here demonstrates that through joint design of both algorithm and architecture, improvements in power, performance and coding efficiency can be achieved.

In particular, MP-CABAC is able to deliver bin-rates up to 3026 Mbin/s by leveraging two highly parallel algorithms, Syntax Element Partitions (SEP) and Interleaved Entropy Slices (IES), and using subinterval reordering to reduce the critical path delay. These algorithms can be easily mapped to parallel hardware while at the same time reducing the area cost and memory bandwidth. Finally, the proposed architecture-driven algorithms maintain high coding efficiency which is critical for the next generation video coding standard.

Acknowledgment

The authors are grateful to M. Budagavi, D. Buss, D. Finchelstein, and A. Wang for their support and valuable feedback. The authors would also like to thank Texas Instruments for algorithm support.

References

[1] 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, Jan. 2010.

[2] 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.

[3] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[4] T.-D. Chuang, P.-K. Tsung, L.-M. C. P.-C. Lin, T.-C. Ma, Y.-H. Chen, and L.-G. Chen, "A 59.5 scalable/multi-view video decoder chip for quad/3D full HDTV and video streaming applications," in *2010 IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2010, pp. 330–331.

[5] J.-W. Chen and Y.-L. Lin, "A high-performance hardwired CABAC decoder for ultra-high resolution video," *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1614–1622, Aug. 2009.

[6] P. Zhang, D. Xie, and W. Gao, "Variable-bin-rate CABAC engine for H.264/AVC high definition real-time decoding," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 3, pp. 417–426, Mar. 2009.

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

[8] S. Nomura, F. Tachibana, T. Fujita, C. K. Teh, H. Usui, F. Yamane, Y. Miyamoto, C. Kumtornkittikul, H. Hara, T. Yamashita, J. Tanabe, M. Uchiyama, Y. Tsuboi, T. Miyamori, T. Kitahara, H. Sato, Y. Homma, S. Matsumoto, K. Seki, Y. Watanabe, M. Hamada, and M. Takahashi, "A 9.7 mW AAC-decoding, 620 mW H.264 720 p 60 fps decoding, 8-core media processor with embedded forward-body-biasing and power-gating circuit in 65 nm CMOS technology," in *2008 IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2008, pp. 262–612.

[9] D. Zhou, J. Zhou, X. He, J. Zhu, J. Kong, P. Liu, and S. Goto, "A 530 Mpixels/s 4096 $\times$ 2160@60 fps H.264/AVC high profile video decoder chip," *IEEE J. Solid-State Circuits*, vol. 46, no. 4, pp. 777–788, Apr. 2011.

[10] D. Marpe and T. Wiegand, "A highly efficient multiplication-free binary arithmetic coder and its application in video coding," in *Proc. 2003 IEEE Int. Conf. Image Processing*, Sep. 2003, vol. 2, pp. II-263–II-266, Vol. 3.

[11] V. Sze and A. P. Chandrakasan, "Joint algorithm-architecture optimization of CABAC to increase speed and reduce area cost," in *Proc. 2011 IEEE Int. Conf. Acoustics, Speech and Signal Processing*, May 2011, pp. 1577–1580.

[12] Recommendation ITU-T H.264: Advanced video coding for generic audiovisual services, ITU-T, Tech. Rep., 2003.

[13] J. L. Mitchell and W. B. Pennebaker, "Optimal hardware and software arithmetic coding procedures for the Q-Coder," *IBM J. Res. & Dev.*, vol. 32, no. 6, pp. 727–736, Nov. 1988.

[14] HEVC test model, HM-2.0. [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/

[15] F. Bossen, "JCTVC-D600: Common test conditions and software reference configurations," Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jan. 2011.

[16] V. Sze, M. Budagavi, and A. Chandrakasan, "VCEG-AL21: Massively parallel CABAC," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), Jul. 2009.

[17] KTA reference software, kta2.7. [Online]. Available: http://iphome.hhi.de/suehring/tml/download/KTA/

[18] V. Sze and A. P. Chandrakasan, "A high throughput CABAC algorithm using syntax element partitioning," in *Proc. 2009 IEEE Int. Conf. Image Processing*, Nov. 2009, pp. 773–776.

[19] D. Finchelstein, V. Sze, and A. Chandrakasan, "Multicore processing and efficient on-chip caching for H.264 and future video decoders," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 11, pp. 1704–1713, Nov. 2009.

[20] T. Tan, G. Sullivan, and T. Wedi, "VCEG-AE010: Recommended simulation common conditions for coding efficiency experiments, rev. 1," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), Jan. 2007.

[21] E. Fleming, C.-C. Lin, N. Dave, A. G. Raghavan, and J. Hicks, "H.264 decoder: A case study in multiple design points," in *Proc. Formal Methods and Models for Co-Design (MEMOCODE)*, Jun. 2008, pp. 165–174.

[22] D. F. Finchelstein, V. Sze, M. E. Sinangil, Y. Koken, and A. P. Chandrakasan, "A low-power 0.7-V H.264 720 p video decoder," in *Proc. 2008 IEEE Asian Solid State Circuits Conf. (A-SSCC)*, Nov. 2008, pp. 173–176.

[23] J. Zhao and A. Segall, "COM16-C405: Entropy slices for parallel entropy decoding," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), Apr. 2008.

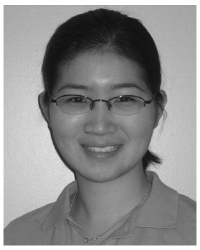
[24] C. Gordon, F. Henry, and S. Pateux, "JCTVC-F274: Wavefront parallel processing for HEVC encoding and decoding," Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jul. 2011.

[25] G. Clare, F. Henry, and S. Pateux, "JCTVC-F275:Wavefront and CABAC flush: Different degrees of parallelism without transcoding," Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jul. 2011.

[26] G. Sullivan and J.-R. Ohm, "JCTVC-F_Notes_dA: Meeting report of the sixth meeting of the joint collaborative team on video coding (JCT-VC), Torino, IT, 14–22 July 2011," Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jul. 2011.

[27] V. Sze and A. P. Chandrakasan, "A highly parallel and scalable CABAC decoder for next-generation video coding," in *2011 IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2011, pp. 126–127.

[28] V. Sze, "Parallel algorithms and architectures for low power video decoding," Ph.D. dissertation, Massachusetts Inst. Technol. (MIT), Cambridge, MA, Jun. 2010.



Vivienne Sze (S'04–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. degree in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 2006 and 2010, respectively. She received the Jin-Au Kong Outstanding Doctoral Thesis Prize, awarded for the best Ph.D. thesis in electrical engineering in 2011.

Since September 2010, she has been a Member of Technical Staff in the Systems and Applications R&D Center at Texas Instruments (TI), Dallas, TX, where she designs low-power algorithms and architectures for video coding. She also represents TI at the international JCT-VC standardization body developing HEVC, the next generation video coding standard. Within the committee, she is the primary coordinator of the core experiment on coefficient scanning and 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 Natural Sciences and Engineering Research Council of Canada (NSERC) Julie Payette fellowship in 2004, the NSERC Postgraduate Scholarships in 2005 and 2007, and the Texas Instruments Graduate Woman's Fellowship for Leadership in Microelectronics in 2008.

Anantha P. Chandrakasan (M'95–SM'01–F'04) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 1989, 1990, and 1994, respectively.

Since September 1994, he has been with the Massachusetts Institute of Technology, Cambridge, where he is currently the Joseph F. and Nancy P. Keithley Professor of Electrical Engineering. Since July 2011, he has been the Head of the MIT EECS Department. His research interests include micro-power digital and mixed-signal integrated circuit design, wireless microsensor system design, portable multimedia devices, energy efficient radios. and emerging technologies. He is a co-author of *Low Power Digital CMOS Design* (Kluwer Academic Publishers, 1995), *Digital Integrated Circuits* (Pearson Prentice-Hall, 2003, 2nd edition), and *Sub-threshold Design for Ultra-Low Power Systems* (Springer 2006). He is also a co-editor of *Low Power CMOS Design* (IEEE Press, 1998), *Design of High-Performance Microprocessor Circuits* (IEEE Press, 2000), and *Leakage in Nanometer CMOS Technologies* (Springer, 2005).

Dr. Chandrakasan was a co-recipient of several awards including the 1993 IEEE Communications Society's Best Tutorial Paper Award, the IEEE Electron Devices Society's 1997 Paul Rappaport Award for the Best Paper in an EDS publication during 1997, the 1999 DAC Design Contest Award, the 2004 DAC/ISSCC Student Design Contest Award, the 2007 ISSCC Beatrice Winner Award for Editorial Excellence and the ISSCC Jack Kilby Award for Outstanding Student Paper (2007, 2008, 2009). He received the 2009 Semiconductor Industry Association (SIA) University Researcher Award. He has served as a technical program co-chair for the 1997 International Symposium on Low Power Electronics and Design (ISLPED), VLSI Design'98, and the 1998 IEEE Workshop on Signal Processing Systems. He was the Signal Processing Sub-committee Chair for ISSCC 1999–2001, the Program Vice-Chair for ISSCC 2002, the Program Chair for ISSCC 2003, the Technology Directions Sub-committee Chair for ISSCC 2004–2009, and the Conference Chair for ISSCC 2010–2011. He is the Conference Chair for ISSCC 2012. He was an Associate Editor for the IEEE JOURNAL OF SOLID-STATE CIRCUITS from 1998 to 2001. He served on SSCS AdCom from 2000 to 2007 and he was the meetings committee chair from 2004 to 2007.