A Deeply Pipelined CABAC Decoder for HEVC Supporting Level 6.2 High-tier Applications

Citation: Chen, Yu-Hsin, and Vivienne Sze. “A Deeply Pipelined CABAC Decoder for HEVC Supporting Level 6.2 High-Tier Applications.” IEEE Trans. Circuits Syst. Video Technol. (2014): 1–1.

Published Version: http://dx.doi.org/10.1109/TCSVT.2014.2363748

Publisher: Institute of Electrical and Electronics Engineers (IEEE)

Permanent Link: http://hdl.handle.net/1721.1/92837

Version: Author's final manuscript: final author’s manuscript post peer review, without publisher’s formatting or copy editing

A Deeply Pipelined CABAC Decoder for HEVC Supporting Level 6.2 High-tier Applications

Yu-Hsin Chen, *Student Member, IEEE,* and Vivienne Sze, *Member, IEEE*

***Abstract*—High Efficiency Video Coding (HEVC) is the latest video coding standard that specifies video resolutions up to 8K Ultra-HD (UHD) at 120 fps to support the next decade of video applications. This results in high-throughput requirements for the context adaptive binary arithmetic coding (CABAC) entropy decoder, which was already a well-known bottleneck in H.264/AVC. To address the throughput challenges, several modifications were made to CABAC during the standardization of HEVC. This work leverages these improvements in the design of a high-throughput HEVC CABAC decoder. It also supports the high-level parallel processing tools introduced by HEVC, including tile and wavefront parallel processing. The proposed design uses a deeply pipelined architecture to achieve a high clock rate. Additional techniques such as the state prefetch logic, latched-based context memory, and separate finite state machines are applied to minimize stall cycles, while multi-bypass-bin decoding is used to further increase the throughput. The design is implemented in an IBM 45nm SOI process. After place-and-route, its operating frequency reaches 1.6 GHz. The corresponding throughputs achieve up to 1696 and 2314 Mbin/s under common and theoretical worst-case test conditions, respectively. The results show that the design is sufficient to decode in real-time high-tier video bitstreams at level 6.2 (8K UHD at 120 fps), or main-tier bitstreams at level 5.1 (4K UHD at 60 fps) for applications requiring sub-frame latency, such as video conferencing.**

***Index Terms*—CABAC, High Efficiency Video Coding (HEVC), H.265, Video Compression**

I. Introduction

HIGH Efficiency Video Coding (HEVC), developed by the Joint Collaborative Team on Video Coding (JCT-VC) as the latest video compression standard, was approved as an ITU-T/ISO standard in early 2013 [1]. HEVC achieves 2× higher coding efficiency than its predecessor H.264/AVC, and supports resolutions up to 4320p, or 8K Ultra-HD (UHD) [2]. It is expected that HEVC will serve as the mainstream video coding standard for the next decade.

HEVC uses context adaptive binary arithmetic coding (CABAC) as the sole entropy coding tool to achieve its high coding efficiency [3]. The superior performance of CABAC is achieved by the following two coding steps (as in the order of encoding, which is the reverse of decoding): it first maps the video syntax elements into its unique binary representations, called *bins*, and then compresses the bins into the bitstream using arithmetic coding with adaptive contexts.

The authors are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA (e-mail: {yhchen, sze}@mit.edu).

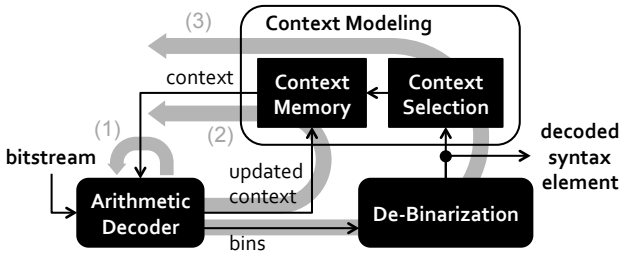


Fig. 1: The serial data dependencies caused by the feedback loops within the CABAC decoding flow. The arrows denote that the decoding of a current bin might depend on its previous bin for (1) the arithmetic decoder state, (2) the updated context, and (3) the selection of the context (or simply bypass).

CABAC is, however, also a well-known throughput bottleneck in H.264/AVC codecs. While high throughput entropy *encoding* has already been demonstrated for HEVC [4], high-throughput *decoding* still remains a challenge. This is due to the highly serial data dependencies caused by several feedback loops within the decoding flow as shown in Fig. 1. This makes it difficult to support the growing demand for higher resolutions and higher frame rates. Also, limited throughput restricts the trade-off for power saving using voltage scaling. As more and more video codecs reside on mobile devices, it becomes a critical concern for battery life.

Efforts have been made to revise the CABAC in HEVC with many throughput-aware improvements while maintaining the coding efficiency [5]. This work will demonstrate an architecture that can maximize the impact of these new features in HEVC including reduced context-coded bins, grouping of bypass bins and reduced memory size requirement. These changes to CABAC in HEVC, however, also create new design challenges. For example, the truncated rice binarization process gives rise to higher bin-to-bin dependencies on the syntax element *coeff_abs_level_remaining* due to the need to parse *cRiceParam*. This makes the parallelization of syntax parsing more difficult. In addition to the improved CABAC performance, HEVC further introduces two high-level parallel processing tools to make the whole decoder more parallelizable, namely tile and wavefront parallel processing (WPP). This work also provides full support for these tools.

Previous works on the CABAC decoder, mostly for the H.264/AVC standard, attempt to increase throughput using mainly two low-level hardware parallelization methods: pipelining and multi-bin per cycle decoding. Pipelining is an effective way of extending parallelism at the temporal domain. However, tight feedback loops at the bin level make

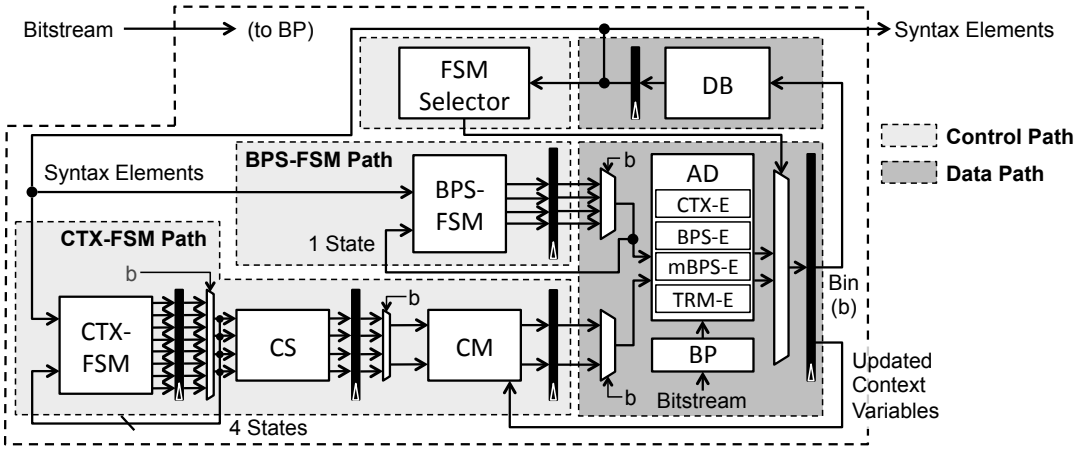


Fig. 2: Block diagram of the CABAC decoder for HEVC. Black-filled blocks represent the stage registers used for pipelining.

the pipelined architecture suffer from an excessive number of stalls [6], [7]. Multi-bin per cycle decoding explores the parallelism by adding additional decoding logic. Many designs decode up to two bins per cycle [8], [9], [10], [11], [12], and a few others reach for more [13], [14]. However, multi-bin per cycle decoding comes at the cost of decreased clock rate. For either of the two parallelization methods to be effective, the decoder needs to support the prefetching of extra decoding information, including the decoding states and context models due to dependencies. Besides prefetching all possibilities as adopted by most of the above works, an alternative scheme is prediction-based decoding, which only speculates the most probable candidate for each bin to be decoded in order to save the hardware overhead [12], [15]. Nevertheless, it lowers the throughput due to the incorrect speculation penalty and increased critical path delay. A highly parallel version of CABAC is presented in [16], which achieves a throughput above 3 Gbin/s through co-optimization of the coding algorithm and hardware architecture; however, the resulting implementation is not standard compliant.

This work proposes an architecture for the CABAC decoder in HEVC with the goal of achieving the highest possible throughput in terms of bins per second. The performance will be optimized toward high bit-rate use cases where high-throughput requirements are critical. Section II and III introduce the techniques to exploit the parallelism for a high-throughput decoder. Specifically, it describes and analyzes the design choice of a deeply pipelined architecture. This architecture incorporates features such as the state prefetch logic, latch-based context memory and separate finite state machines to minimize stalls, and employs a multi-bypass-bin decoding scheme to further increase throughput. Section IV presents the experimental and analytical results under the common and theoretical worst-case conditions, respectively. The synthesized throughput, area and power will also be reported. The performance of the high-level parallel processing tools that enable running multiple CABACs per frame are discussed in Section V.

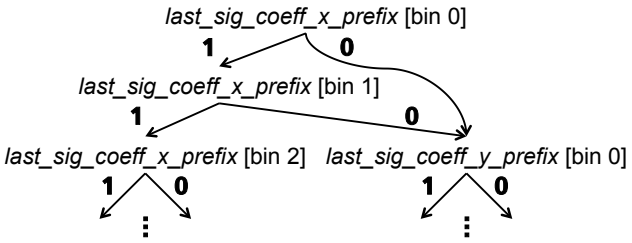
II. Proposed CABAC Decoder Architecture

To realize the CABAC decoder for HEVC with the highest possible throughput measured in bins per second, we seek to increase two key factors, namely clock rate (cycles per second) and average number of decoded bins per clock cycle. The proposed design features a deeply pipelined architecture to achieve a high clock rate. This section will describe the pipeline design in detail. Timing analysis on each of the pipeline stages will also be provided to demonstrate its impact on increasing the clock rate.

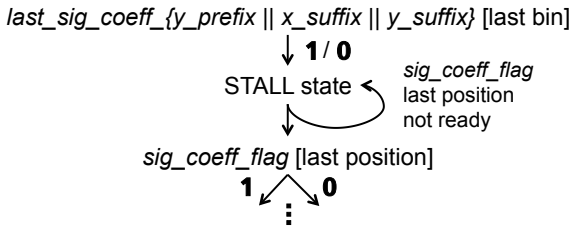
A. Architecture Overview

Fig. 2 illustrates the block diagram of the proposed CABAC decoder architecture. The bitstream parser (BP) buffers the incoming bitstream and feeds the arithmetic decoder (AD) according to the AD decoding mode. There are four decoding modes, which invoke four different decoding processes: context-coded bin decoding (CTX), bypass bin decoding (BPS), multi-bypass-bin decoding (mBPS) and terminate bin decoding (TRM). With the request of a decoding process, the corresponding decoding engine (CTX-E, BPS-E, mBPS-E or TRM-E) would be activated to perform the operation. The decoded bins are then reassembled at the de-binarizer (DB) into syntax elements. The rest of the decoder is responsible for gathering decoding information for AD. First, the decoding mode at each cycle is determined by two finite state machines (FSM), BPS-FSM and CTX-FSM, according to the HEVC-compliant decoding syntax. Only one out of the two FSMs controls AD within a single cycle, which is decided by the FSM Selector based on previously decoded syntax elements. In addition, if the decoding mode invokes the CTX process, the estimation of bin probability as modeled by the context variables (CVs) is also required. CVs are initialized and stored in the context memory (CM), and the required one for decoding is retrieved by the context selector (CS). CS is only controlled by CTX-FSM. After the CTX process, the updated CV is written back to CM for future access.

Among the decoding engines in AD, CTX-E dominates the decoding complexity and contains the critical path of



(a)



(b)

Fig. 3: Different parts of the binary decision tree (BDT) for the FSM prefetch logic. (a) A fully expanded BDT avoids the need for stall cycles. (b) Stalls are kept to avoid creating dozens of extra states, which would increase the critical path delay of the FSMs.

AD. Therefore, we optimize CTX-E using the techniques introduced in [16], including leading-zero detection, subinterval reordering, early range shifting and next cycle offset renormalization, for a 22% reduction in delay.

B. Deep Pipelining with State Prefetch Logic

The architecture discussed in Section II-A is pipelined for high-throughput decoding. The pipeline stages are shown in Fig. 2, in which the black-filled blocks represent the stage registers. The function blocks are divided into two groups, the data path and the control path. The data path consists of two pipeline stages: AD and DB. The control path is further divided into two sub-paths based on the two FSMs. The BPS-FSM path only has one stage that controls the data path directly, while the CTX-FSM has a deeper three-stage pipeline, including CTX-FSM, CS and CM. The overall design is a deeply pipelined five-stage structure. If the next decoder state depends on the decoded syntax element of the current state, this architecture could impose up to four stall cycles.

State prefetch logic is introduced to eliminate the majority of stalls imposed by the tight dependencies described above. Fig. 3a shows an example of how the prefetch logic works. Based on the binary value of the decoded bin at the current decoder state, there are two choices for the next state. As in the case of the syntax element *last_sig_coeff_x_prefix*, if its first bin is 1, the decoding will continue to its second bin; otherwise, the decoding will jump to the first bin of *last_sig_coeff_y_prefix*. The next state logic of CTX-FSM will prefetch both of the possible states and pass both of them along the pipeline. The decision of which next state out the two that will get executed at AD is delayed until the current bin is decoded. Following this manner, the FSM logic becomes a binary decision tree (BDT). However, the construction of BDT is a trade-off between number of states and number of stalls. If the BDT is fully expanded for all possibilities, the number of states would grow exponentially and increase the critical path delay. To balance between the two aspects, the BDT is optimized to eliminate most of the throughput-critical stalls while keeping the number of states to a minimum. Based on the analysis of common video bitstreams, the transform coefficients account for a large proportion of the total number of bins, and its decoding has a significant impact on the throughput of CABAC [5]. Therefore, its related parts of BDT, including syntax elements *sig_coeff_flag*, *coeff_abs_level_greater1_flag*, *coeff_abs_level_greater2_flag*, *coeff_sign_flag* and *coeff_abs_level_remaining*, are fully expanded, while the rest of BDT is optimized to avoid creating an excess number of states for each syntax element. Fig. 3b shows an example where the stalls are kept. The FSM stalls until the position of the last significant coefficient being decoded, so it can select the CVs for syntax elements *sig_coeff_flag*. In the worst case, the decoder will stall for three clock cycles for a transform block. Nevertheless, if the states were to be fully expanded, dozens of unique states need to be created to account for the different combinations of *last_sig_coeff_x_prefix*, *last_sig_coeff_y_prefix*, *last_sig_coeff_x_suffix* and *last_sig_coeff_y_suffix*. This would add an extra 10% to 15% more states to the BDT, increasing the critical path of the FSMs. The overall throughput degradation due to the remaining stalls is approximately 12% (tested with bitstream *Nebuta* at QP of 22, with techniques discussed in Section III applied). The syntax element that contributes the most stalls is *coded_sub_block_flag*, which results in 6% throughput degradation due to the bin-to-bin dependency. The stall example in Fig. 3b also contributes 2%.

The number of possible next states at each pipeline stage grows exponentially with the depth of the pipeline. To resolve one state at the AD stage for decoding requires CTX-FSM to compute next eight possible states at every cycle since it occurs three stages before AD. Although BPS-FSM only has the control path depth of one, it still needs to compute next four possible states due to the multi-bypass-bin decoding scheme, which will be discussed in Section III-B. At each pipeline stage, the bins decoded by AD at previous cycle are used to select the correct inputs states of the current cycle from all input states, as shown by the mux at the beginning of stage CS, CM and AD in Fig. 2.

C. Pipeline Stages Timing Analysis

The pipeline stages as shown in Fig. 2 are optimized to achieve the highest clock rate. Table I shows the critical path delay of the combinational logic in each stage of the pipeline. Stage AD as well as CM has the largest delay in this architecture and therefore determines the performance of the entire CABAC decoder. Within stage AD, CTX-E dominates the logic delay among all four decoding engines. The block diagram of CTX-E and BPS-E are illustrated in Fig. 4a and Fig. 4b, respectively, to show the critical path. The critical path

Stage Name	Critical Path Delay (ns)
CTX-FSM	0.42
CS	0.26
CM	0.44
AD	0.44
DB	0.41
BPS-FSM	0.20
FSM Selector	0.23

TABLE I: The critical path delay of each stage in Fig. 2 at synthesis level. A 45nm SOI process is used. The delay is calculated with only the combinational logic considered.

of CTX-E starts from input *stateIdx* and ends at output *range*. Its critical path delay is approximately 300 ps in a 45nm SOI process. The critical path of BPS-E starts from input *shift* and ends at output *offset*. Its critical path delay is around 170ps. The small delay difference between stage AD and CTX-FSM explains that the effort put to optimize the CTX-FSM BDT is as important as optimizing the delay of the CTX-E engine in AD. The five-stage pipeline provides equally distributed delays across stages, which enable high clock rate for high CABAC decoding throughput.

This timing information could be applied to different designs for comparisons from an architecture point of view. An architecture adopted in previous works that achieves high performance is the two-stage pipeline two-bin per cycle decoder [8], [9]. Its first stage consists of CS and CM, and its second stage consists of a two-bin AD, DB and FSM (the syntax element parser in [9]). The critical path lies on the second stage. To translate the timing requirement of this architecture for comparison, it should first be noted that the delay of the two-bin AD, as timing optimized in [9], is about 1.3× higher than the proposed optimized one-bin AD. Also, it is possible to co-optimize the delay of DB and FSM. Under an optimistic assumption that only the delay from stage DB is considered, the total delay of this stage as well as the critical path of the entire architecture is approximately

$$1.3 \times delay(AD) + delay(DB), \tag{1}$$

which is 0.98 ns. Recall that the overall throughput is the product of clock rate and number of decoded bins per cycle. Thus, to achieve the same throughput as the five-stage pipeline in this paper, the average decoded bins per cycle of the architecture in [9] needs to be at least 2.2× higher. In addition, as indicated by the analysis in Section II-D, the proposed architecture requires less area.

D. Pipelining vs. Multi-Bin per Cycle Processing

Pipelining and multi-bin per cycle processing are the two low-level parallelization methods, as introduced in Section I, used to speed up the processing of CABAC decoding. This work applies both of these method in the form of a deeply pipelined architecture with a multi-bypass-bin decoding scheme (see Section III-B). Many previous works also combine the two and optimize for their best throughput. It becomes an important design consideration to understand the implication of the two methods from an implementation point of view for CABAC in HEVC.

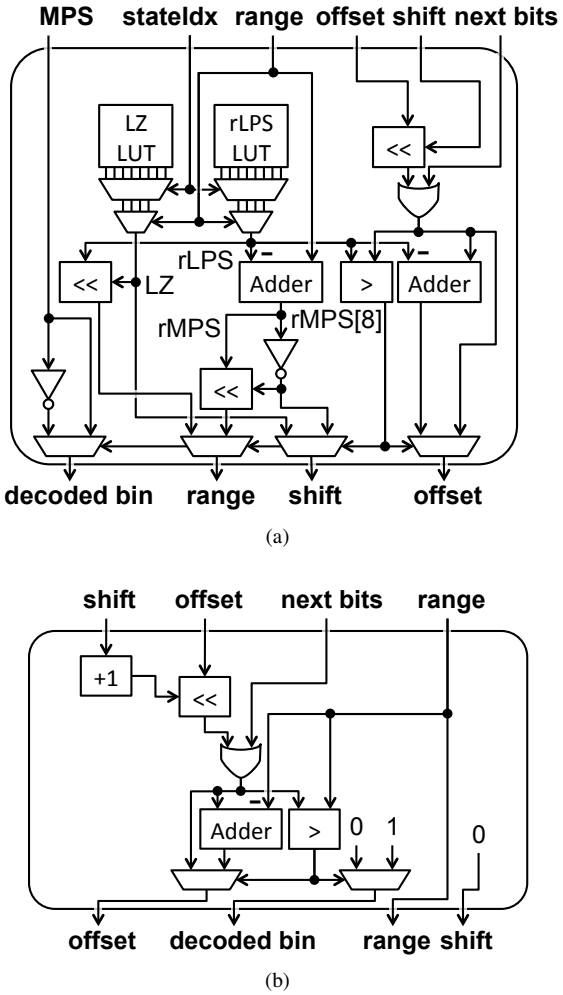


Fig. 4: Block diagram of (a) CTX-E and (b) BPS-E.

Ideally, a design capable of processing N bins per cycle could deliver similar performance to a N-stage pipelined one. Without considering the design complexity and cost, the parallelism can be fully exposed to achieve the throughput speedup of at most N times by expanding the FSM BDT. The former can hide the logic delay of the N-bin AD by performing parallel pre-computation, and the latter can reach the clock rate of N times faster than without pipeline. In reality, however, the cost to fully expose the parallelism is too high to be practical. This leads to the trade-off in BDT design discussed in Section II-B for both methods, and the N-bin per cycle design may further suffer from reduced clock rate since the serial nature of AD computation limited the efficacy of parallel pre-computation. In addition, for the control path that decides the decoding state and CVs, both methods are required to resolve 2^N possibilities. The pipelined architecture can reduce the number of candidates by half for each of the following stages, lowering the hardware cost for blocks such as CS and CM, whereas the N-bin per cycle design needs to pass all of the 2^N possible candidates to the data path. Also, AD in the N-bin per cycle design has to handle all possible N-bin combinations of context-coded, bypass and terminate bins. The number of possible combinations grows exponentially, so it is

only feasible to speed up some of the most common sequences for N larger than two, which becomes another trade-off with reduced clock rate.

In order to go for high parallelism, a deeply pipelined architecture is more plausible than a deep multi-bin design from the above analysis. It is possible to divide the parallelism N in the way that $N = N_1 N_2$, where N_1 and N_2 are the number of pipeline stages and number of multi-bin per cycle, respectively. With smaller N_1 and N_2, it is easier to take advantage of both methods as well as the throughput improvement features introduced by HEVC that benefit the multi-bin design, such as grouping of bypass bins and grouped CVs. In this work, we use the deeply pipelined architecture to increase the clock rate that can benefit all types of bins, and employ the multi-bin processing only on the bypass bins to further speed up the throughput demanding bitstreams without affecting the clock rate.

E. Latch-Based Context Memory

The state prefetch logic addresses the stall issue for the deeply pipelined architecture at the cost of higher computational requirements to the hardware. This concern is most significant for CM, where two context variables need to be prefetched as CM occurs in the stage before AD. Conventional architectures use SRAM for low area and low power memory access. Some works implement extra caches to achieve multiple reads and writes within the same cycle [6], [9]. But these designs cannot support truly random high-bandwidth memory access as required by the state prefetch logic due to the limited cache size. Fortunately, HEVC has 3× fewer contexts than H.264/AVC. Thus, the required space to store all CVs reduces to 1 kbit, which makes the all-cache CM implementation a more practical option.

Latch-based cache requires smaller area and consumes less power when compared to register-based cache. To ensure glitch-free latch access, Fig. 5 demonstrates the design of the latch enable signal with its timing diagram. The signal coming into the enable (EN) port of the latch is constrained to settle within the first half of the clock cycle; thus the logic delay of the Enable Decoder, which is the grey region of the signal ED_N in the timing diagram, needs to be smaller than half of the clock cycle. EN always resets to low when clock is high, and will only be high at the second half of the clock cycle. Although only half of the cycle is available for the logic delay of the Enable Decoder and for the write data (wData) to update the latch, it is not an issue for the deeply pipelined architecture, as both the write address (wAddr) and wData signals are coming from the pipeline stage registers. The timing requirement can be easily met.

Table II lists the comparison of area and power between all three possible memory implementations at synthesis level. At the size of 1 kbit, the latch-based design takes only 15% more area than SRAM, but reduces power consumption by 1.7×. When compared to the register-based design, the latch-based memory not only reduces power by 2.8×, but also reduces area by 1.5×.

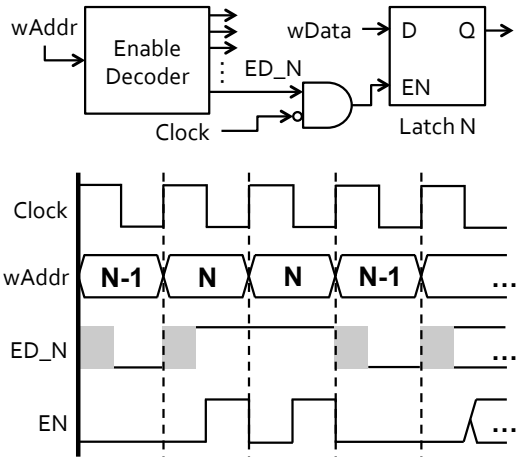


Fig. 5: Glitch-free latch enable design with its timing diagram.

Memory Design	Area (eq. gate count)	Power (mW)
SRAM	11.7k	8.10
Register-based	20.2k	13.10
Latch-based	13.4k	4.68

TABLE II: Comparison between different types of 1 kbit memory implementation for CM at synthesis level. The power is measured under a 2GHz clock and an 100% memory read/write pattern.

III. Throughput Improvement Techniques

In Section II, we introduce a deeply pipelined CABAC decoder architecture that can achieve a high clock rate. The stalls caused by data hazard in the deep pipeline are handled by a conventional state prefetch logic. In this section, we seek to further improve the throughput by applying two architecture techniques that are motivated by the high-throughput features of CABAC in HEVC. One is to reduce more stalls with a shallower pipeline for bypass bins. The other one will equip the architecture with multi-bin processing capability. It will also talk about the essential hardware changes required to support the high-level parallel processing tools in HEVC.

A. Separate FSM for Bypass Bins

HEVC has fewer context-coded bins than H.264/AVC, resulting in a larger proportion of bypass bins. In addition, most of the bypass bins are grouped together to reduce the amount of switching between bypass bins and context-coded bins. These observations lead to the design of separate finite state machines for context-coded bins (CTX-FSM) and grouped bypass bins (BPS-FSM). The FSM Selector is used to select the FSM that should be enabled for each cycle. CTX-FSM can handle all decoding modes except the mBPS mode, while BPS-FSM only operates for the grouped bypass bins with the BPS or mBPS mode. BPS-FSM does not manage all bypass bins, however, as that complicates the logic in the FSM Selector to switch between the two FSMs frequently, creating a longer critical path.

As discussed in Section II-B, the CTX-FSM path is divided into three stages to support CS and CM while maintaining

Syntax Hierarchy	Syntax Elements
Prediction Unit	*mpm_idx*, *rem_intra_luma_pred_mode*
Transform Unit	*last_significant_coeff_x_suffix*, *last_significant_coeff_y_suffix* *coeff_sign_flag*, *coeff_abs_level_remaining*
Loop Filter	*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*

TABLE III: List of syntax elements that utilize the BPS-FSM path.

a short critical path. As CS and CM are not needed for bypass bins, the BPS-FSM path only has one stage. Separating grouped bypass bins out of CTX-FSM simplifies the logic of CTX-FSM. Also, a shallower pipeline for grouped bypass bins eliminates stalls within the bypass bins. Since the FSM Selector and the two FSMs reside in different pipeline stages, the critical path will not be affected. Table III provides a complete list of syntax elements that utilize the BPS-FSM path. The benefit is most significant for syntax element *coeff_abs_level_remaining*, which can take up around 5% to 10% of the total workload in common bitstreams. Three stall cycles are saved for each *coeff_abs_level_remaining* when compared to without a separate BPS-FSM path since the binarizaton of the next syntax element depends on the value of the current one due to the updating of *cRiceParam*. When tested with high bit-rate bitstreams such as *Nebuta* at QP of 22, where transform coefficients account for a large proportion of the workload, the separate FSM design increases the overall throughput by almost 33%.

B. Multi-Bypass-Bin Decoding

Grouping of bypass bins also increases the benefit of decoding more than one bypass bin per cycle. The logic delay of BPS-E is around half of the optimized CTX-E. Also, concatenating two BPS-E will not double the critical path delay since the critical path in BPS-E is from input *shift* to output *offset* as shonw in Fig. 4b. Therefore, without increasing the AD stage delay, the deeply pipelined architecture can decode two bypass bins per cycle with mBPS-E, which is the concatenation of two BPS-Es. The corresponding decoding mode, mBPS, is used for the bins of *coeff_sign_flag* and *coeff_abs_level_remaining*. Since the number of bins combined from these two syntax elements account for at least 10% to 20% of the total number of bins in common video bitstreams, this scheme can improve the throughput significantly. To support mBPS, the BDT width of the BPS-FSM control path needs to be doubled, as the prefetch logic has to compute next four possible states instead of two.

It is also possible to increase the number of bypass bins decoded per cycle, M, beyond two at the cost of increased complexity of BPS-FSM and possible extra critical path delay. By first ignoring the impact on the critical path delay, Fig. 6 shows the improvements on the average decoded bins per cycle by setting M to 2, 3 and 4 and compare to M of 1. While the improvements vary across different testing sequences and QP configurations, it is clear that increasing M beyond 2 only gives marginal improvements compared to increasing M from 1 to 2. This justifies the design with M equals to 2 considering the extra cost.

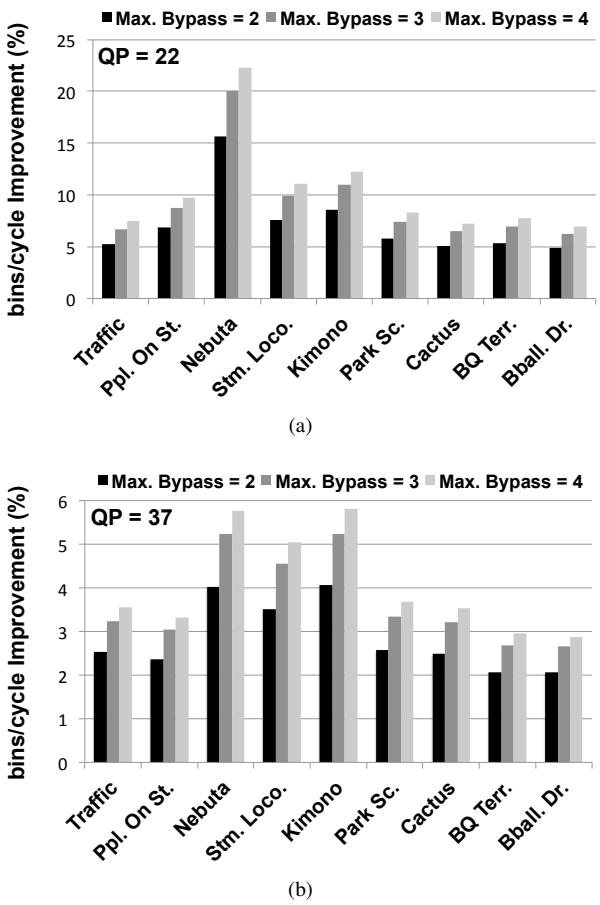


Fig. 6: The improvements on the average decoded bins per cycle by increasing the maximum amount of bypass bins to be decoded within the same cycle. All Class A and B common testing sequences in [17] are tested. The results from different coding structures (All Intra, Low Delay, Random Access) are averaged within each sequence. (a) and (b) shows the configurations at QP equals to 22 and 37, respectively. The detail of the testing sequences could be found in Table IV.

C. Support of High-Level Parallel Processing Tools

The idea of high-level parallel processing in HEVC, including both the tile processing and WPP, is to divide each video frame into several smaller parts, and all parts can run in parallel by applying the same CABAC decoding process across multiple CABAC decoding engines. To support these tools, additional high-level control flow and CM for WPP are required. In this work, with only one set of CABAC decoding engines, the CM access pattern is as illustrated in the example shown in Fig. 7, and is described as follows:

1) For the decoding of coding tree units (CTUs) in the first row, the decoder retrieves and updates the CVs from *CV Set 1* in CM. A *CV Set* contains all required CVs for the decoding of a CTU.

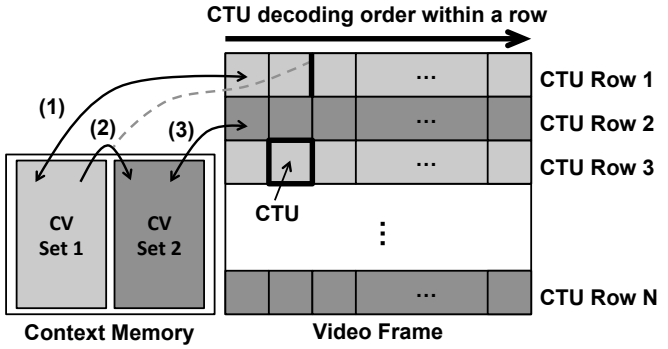


Fig. 7: An example of the CM access pattern with WPP enabled: (1) The CTUs in the first row use and update the CVs from *CV Set 1* in CM. (2) The CVs in *CV Set 1* are replicated into *CV Set 2* after the second CTU in the first row finishes decoding and before the third CTU begins to update it. (3) The CTUs in the second row use and update the CVs from *CV Set 2* in CM.

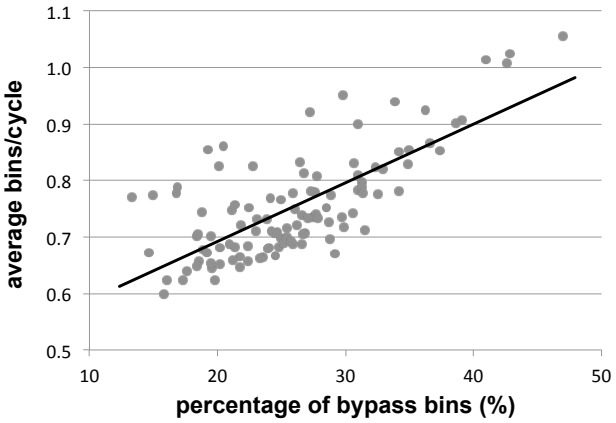


Fig. 8: The linear dependency between the percentage of bypass bins in the bitstream and the average decoded bins per cycle achieved by the proposed design. The data points are collected from the simulation with common test sequences in Table IV.

2) After finishing the decoding of the second CTU in the first row, and before the decoding of the third CTU updates the CVs in *CV Set 1*, CM replicates the CVs from *CV Set 1* to *CV Set 2*.
3) For the decoding of CTUs in the second row, the decoder retrieves and updates CVs from *CV Set 2* in CM.

This process is repeated for every two adjacent CTU rows. Odd number CTU rows use the CVs in *CV Set 1* and replicate it to *CV Set 2*, and even number CTU rows use the CVs in *CV Set 2* and replicate it to *CV Set 1*. The above description suggests that the size of CM needs to be large enough to store two set of CV values. In the case of HEVC, the size of CM becomes 2 kbit. With an all-cache CM design, the delay of the replication process can be greatly shortened than with SRAM since more CVs could be copied between the two sets per clock cycle.

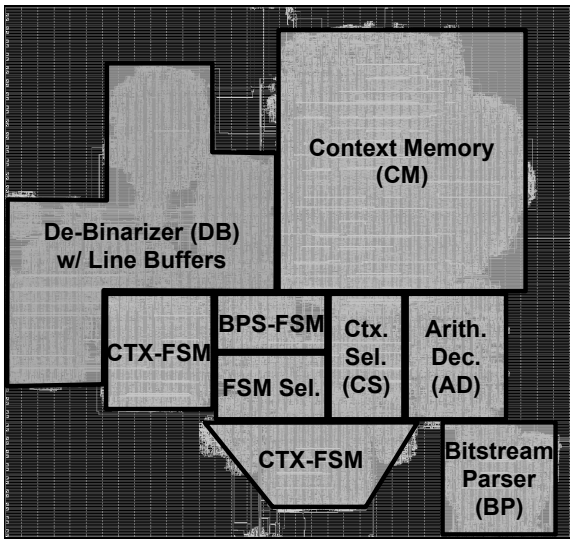


Fig. 9: Layout of the proposed CABAC decoder.

IV. Experimental Results

A. Experimental and Synthesis Results

Table IV shows the simulated decoding performance of the proposed architecture for common test bitstreams [17]. It has taken the impact of stalls (as discussed in Section II-B) and the throughput improvement features (as discussed in Section III) into account. In general, the bins per cycle of the high bit-rate sequences, especially the all-intra (AI) coded ones, is higher than that of the low bit-rate bitstreams. For example, *Nebuta*, with a bit-rate up to 400 Mbps, can be decoded at 1.06 bin/cycle.

The variation of the decoding performance is due to the design trade-offs of the deeply pipelined architecture. Since more efforts are put into speeding up the processing of transform coefficients, the high-demanding bitstreams that have high bit-rate and consist of a large proportion of transform coefficient bins will benefit more from the design. These bins are mostly bypass bins. Specifically, as shown in Fig. 8, there is a clear linear dependency between the percentage of bypass bins in the bitstreams and the average decoded bins per cycle achieved by the design. This suggests that the proposed design is suitable for processing high-demanding bitstreams in HEVC, and the performance only scales back for the less demanding bitstreams, which have lower throughput requirements.

The design is implemented in an IBM 45nm SOI process with 0.9V supply voltage. At synthesis level, it achieves a maximum clock rate of 1.9 GHz [18]. After place-and-route, the maximum clock rate becomes 1.6 GHz (with 30 ps clock uncertainty margin). A snapshot of the layout is shown in Fig. 9. For the AI-coded *Nebuta* bitstream at QP of 22, the throughput reaches 1696 Mbin/s, which is already sufficient for the real-time decoding of level 6.2 (8K UHD at 120 fps) video bitstreams. The total gate count of the CABAC decoder is 92.0k and 132.4k at synthesis and place-and-route levels, respectively. In order to support WPP, the size of

Class	Sequence	Frame Rate (Hz)	QP	Coding Structure	Bit Rate (Mbps)	Bypass Bins (%)	Bins/Cycle
A (2560x1600)	Traffic	30	22	AI	101.89	34.9	0.83
				LD	13.83	18.4	0.70
				RA	13.24	23.8	0.73
			37	AI	18.53	26.9	0.71
				LD	1.00	17.7	0.64
				RA	1.34	22.4	0.66
	People On Street	30	22	AI	104.72	37.4	0.85
				LD	37.56	27.6	0.78
				RA	32.83	30.9	0.78
			37	AI	20.40	29.2	0.67
				LD	5.02	25.9	0.69
				RA	4.64	28.8	0.70
	Nebuta	60	22	AI	403.02	46.9	1.06
				LD	239.32	42.8	1.02
				RA	216.38	41.1	1.01
			37	AI	81.55	26.4	0.83
				LD	9.06	13.3	0.77
				RA	7.14	16.9	0.79
	Steam Locomotive	60	22	AI	100.39	36.2	0.92
				LD	30.52	20.1	0.82
				RA	23.55	22.8	0.82
			37	AI	14.54	27.3	0.78
				LD	1.20	19.6	0.64
				RA	1.21	21.8	0.66
B (1920x1080)	Kimono	24	22	AI	22.25	38.6	0.90
				LD	5.21	27.8	0.81
				RA	4.80	32.3	0.82
			37	AI	3.83	28.9	0.77
				LD	0.58	20.2	0.68
				RA	0.55	25.0	0.70
	Park Scene	24	22	AI	52.75	34.9	0.85
				LD	7.97	19.5	0.70
				RA	7.69	26.1	0.75
			37	AI	7.31	24.5	0.71
				LD	0.59	17.4	0.62
				RA	0.73	21.7	0.65
	Cactus	50	22	AI	105.39	30.7	0.83
				LD	20.05	18.8	0.74
				RA	18.44	22.5	0.75
			37	AI	14.30	26.6	0.69
				LD	1.29	19.5	0.65
				RA	1.40	23.6	0.66
	BQ Terrace	60	22	AI	180.18	31.0	0.90
				LD	52.82	15.0	0.77
				RA	39.64	16.8	0.78
			37	AI	21.81	24.8	0.68
				LD	0.79	15.8	0.60
				RA	1.00	19.8	0.62
	Basketball Drive	50	22	AI	71.14	26.7	0.81
				LD	19.86	21.4	0.76
				RA	17.40	24.1	0.77
			37	AI	8.52	23.3	0.66
				LD	1.61	21.3	0.66
				RA	1.50	24.6	0.67

TABLE IV: Simulated decoding performance of the proposed design for common test sequences [17]. Each sequence is coded with two QP configurations and three coding structures: All Intra (AI), Low Delay (LD) and Random Access (RA).

CM is 1 kbit×2 as discussed in Section III-C. The power consumption after place-and-route is 51.6 mW. Table V shows the area and power breakdowns of the function blocks. Among different blocks, CM consumes the largest proportion of area and power (34.6% and 28.9%, respectively). If we replace the latch-based CM with a register-based one, the total area and power consumption would increase by 17.3% and 52.0%, respectively, as suggested by Table II, which justifies the use of latch-based memory design.

Fig. 10 shows the performance comparison between the proposed work and previous designs, including both H.264/AVC and HEVC works. Since the clock rate and number decoded bins per cycle can be regarded as the degree of pipelining[1] and the degree of multi-bin processing, respectively, this plot shows how different works optimize the performance based on the two low-level hardware parallelization methods. It should be noted that while all previous works are reporting synthesis results, we are presenting the result at post-place-and-route level. The bins per cycle number of the proposed work spans across a region since it varies with the testing bitstreams. The highest and lowest numbers in the plot are from Table IV. Though the works spread across the entire space in this plot, the designs using the same or similar technology nodes

[1]if the effects of different technology nodes are compensated.

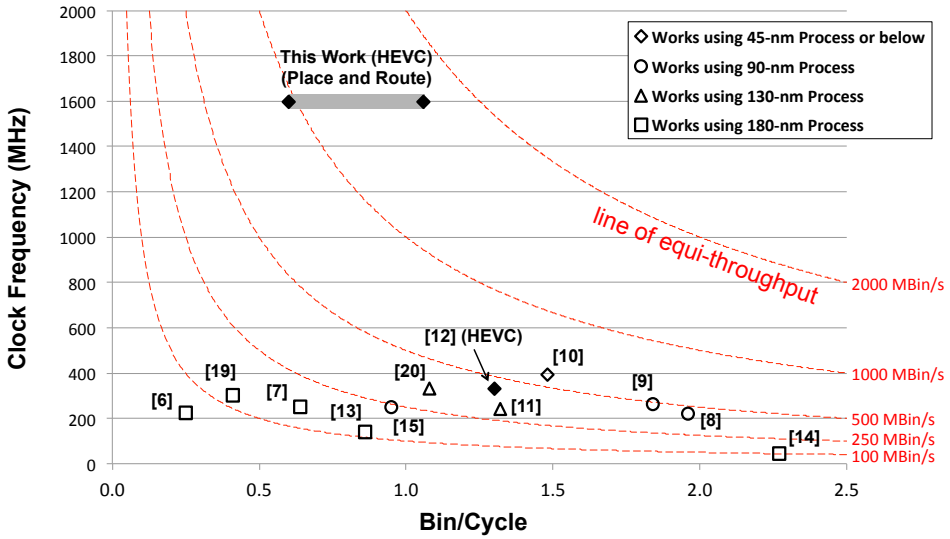


Fig. 10: Performance comparison between the proposed work and previous designs, including both H.264/AVC and HEVC works [19], [6], [7], [13], [14], [11], [20], [8], [15], [9], [10], [12]. The works using the same or similar technology processes are grouped into the same marker. The filled markers denote the works for HEVC, while the rest is for H.264/AVC. It should be noted that while all previous works are reporting synthesis results, the result of our work is obtained after place-and-route. The performance of the proposed design spans across a range since it depends on the testing bitstreams. This plotted range uses the data from Table IV.

	Gate Count	Power (VDD = 0.9V)
Total	132.4k (100%)	51.6mW (100%)
Arithmetic Decoder	7.1%	17.0%
Context Memory (1 kbit×2)	34.6%	28.9%
Finite State Machines (CTX+BPS)	12.5%	15.6%
Line Buffers	17.0%	9.6%
Context Selection	4.7%	6.2%
De-binarization	13.9%	8.4%
Bitstream Parser	8.6%	5.6%

TABLE V: The area and power breakdowns of the proposed design after place-and-route in IBM 45nm SOI process.

usually yield similar throughputs in terms of bins per second. By comparing the works within each of these groups, it shows a more clear picture of how the different architectures translate to performance trade-offs. The result also shows that the proposed design has a clear throughput advantage over previous works. It comes not only from the advance in technology, but also from the architecture techniques used as described in Section II and III. Table VI summaries a more detailed comparison between the proposed design and three recent works [8], [9], [12].

B. Analytical Worst-Case Performance

The decoding latency of CABAC is another important performance indicator. For applications such as video conferencing or live broadcasting, sub-frame latency is required for real-time streaming. In addition, within the design of a HEVC decoder, the syntax elements decoded by CABAC are further processed by the HEVC decoder backend. The backend usually processes data at the granularity of a CTU. Therefore, the latency variation of a CTU determines the buffer size between the CABAC decoder and the backend.

The decoding latency is directly proportional to the bin-rate of the video bitstream. HEVC defines the maximum bin limits at three granularities: within a CTU, within a frame, and across frames. These limits, therefore, correspond to the worst-case decoding latencies of a CTU, a frame, and multiple frames, respectively. According to the equations shown in Appendix A, the worst-case bin-rate limits of the three granularities are listed in Table VII at specific bitstream levels and tiers with given resolutions and frame rates. The calculation uses the updated parameters listed in [22] instead of in [1]. The limits tend to be lower when larger latency is tolerated since workload can be averaged across CTUs and frames. The decoder throughput needs to be higher than these limits to guarantee real-time low latency decoding.

To assess the performance of the proposed CABAC decoder under these worst-case scenarios, the decoder is assumed to decode with the maximum number of bins per CTU as described above for the maximum bin-rate per CTU, and compared to the limits at three different granularities. Taking the number of bypass bins decoded with the mBPS mode as well as the stalls into account, the design in this work can decode at 1.44 bin/cycle. The corresponding throughput is 2314 Mbin/s, which is higher than the one under the common test conditions. The reason is that, under the worst-case scenarios, the decoding is dominated by the bypass bins (5096 bypass bins out of 5960 total bins per CTU, or 85% of bypass bins), and the proposed design is optimized toward the decoding of more bypass bins. In Table VII, we shade in grey all bin-rate limits that can be achieved by this design in real-

		Lin [8]	Liao [9]	Choi [12]	**This Work**
Standard		AVC	AVC	HEVC	HEVC
Technology		UMC 90nm	UMC 90nm	Samsung 28nm	IBM 45nm SOI
Gate Count	synthesis	82.4k	51.3k	100.4k[1] (0.047mm^2)	92.0k
	place & route				132.4k
SRAM Size		N/A	179B	N/A	N/A
Max. Frequency	synthesis	222 MHz	264 MHz	333 MHz	1900 MHz
	place & route				1600 MHz
Bins/Cycle		1.96	1.84[2]	1.30	1.06[3]
Throughput	synthesis	435 Mbin/s	486 Mbin/s	433 Mbin/s	
	place & route				1696 Mbin/s

[1] without the bitstream parser buffer [21]
[2] with the test bitstream bit-rate at 130 Mbps
[3] with the test bitstream bit-rate at 403 Mbps

TABLE VI: Comparison on the results of different CABAC decoder implementations. The gate counts of all four works include CM, either implemented by caches or SRAM.

Level	**4.0**	**4.1**	**5.0**	**5.1**	**5.2**	**6.0**	**6.1**	**6.2**
Frame Height	1080	1080	2160	2160	2160	4320	4320	4320
Frame Width	2048	2048	4096	4096	4096	8192	8192	8192
Frame Rate	30	60	30	60	120	30	60	120
Main Tier								
Per CTU	1550	3100	6200	12400	24800	24800	49600	99200
Per Frame	292	585	813	1270	2540	2540	5070	13000
Multi-Frame	16	27	33	53	80	80	160	320
High Tier								
Per CTU	1550	3100	6200	12400	24800	24800	49600	99200
Per Frame	292	585	1170	2340	4680	4680	9350	18700
Multi-Frame	40	67	133	213	320	320	640	1070

TABLE VII: The worst-case bin-rate (Mbin/s) limits of three granularities with different bitstream levels and tiers at given resolutions and frame rates (fps). The table cells shaded in grey are the achievable throughputs by the proposed design with a single CABAC decoder.

time for each granularity under worst-case scenarios. If multi-frame latency could be tolerated, the design can decode level 6.2 bitstreams in real-time for both main and high tiers. For applications that require sub-frame latency, it is also capable of decoding at level 5.1 (4K UHD at 60 fps) for main tier or level 5.0 (4K UHD at 30 fps) for high tier in real-time.

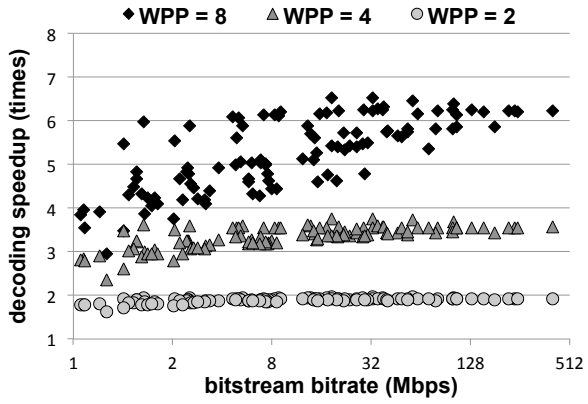
V. High-Level Parallel Processing Tools

HEVC provides two high-level parallel processing tools, WPP and tile processing, to speed up the decoding when multiple CABACs are available. WPP parallelizes the processing of each row of CTUs within a frame. Tile processing divides a frame into several rectangular tiles for parallel processing. While adjacent CTU rows in WPP mode still have CV and line buffer dependencies on each other, the processing of tiles are completely independent for CABAC. Fig. 11 demonstrates the speedup of CABAC decoding performance using both WPP and tile processing. In both cases, the amount of parallelism is defined as the maximum amount of rows or tiles that can be decoded at the same time by duplicating the decoding hardware of the proposed design. The testing sequences are the common sequences as listed in Table IV. For each data point, the speedup is compared with using the same configuration but without any high-level parallelism.

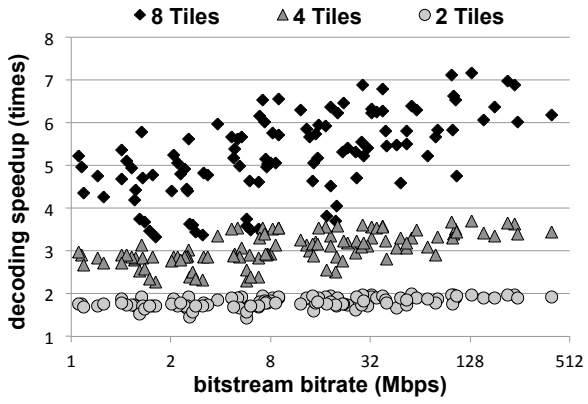
Due to the dependency between CTU rows in WPP, and the workload mismatch of the tiles in tile processing, the performance speedup through high-level parallelism is not linear. There is a clear trend in the case of WPP that bitstreams with higher bit-rate get more consistent speedup than those with lower bit-rate. In the case of tile processing, it shows less speedup saturation when increasing the bit-rate. These observations could be explained by the following analysis. Fig. 12 demonstrates the histogram of required decoding cycles per CTU by the proposed design for two different bitstreams. The AI-coded *Nebuta* sequence at QP of 22 represents the high bit-rate and high throughput bitstream, and the LD-coded *BQ Terrace* at QP of 37 is the exact opposite. Table VIII gives the average and the standard deviation σ of the required decoding cycles per CTU for data in Fig. 12. Even though the σ of *BQ Terrace* is much lower than that of *Nebuta*, it is much higher than its own average. In terms of decoding performance in the case of WPP, this results in high uncertainty in CTU dependency for low bit-rate bitstream and contributes to the wide range of speedup performance. The speedup is relatively consistent for high bit-rate bitstream since the σ is only a fraction of the average. For tile processing, the unit of comparison becomes a tile, which is a set of CTUs. At tile level, the variation of the CTU performance is averaged across multiple CTUs and becomes less significant. The performance is affected more by the spatial variation of the contents within a frame, and is dependent on the specific sequence under test.

VI. Conclusion

In this paper, we propose the hardware architecture of a CABAC decoder for HEVC that leverages the throughput improvements of CABAC introduced in HEVC. It also supports the two new high-level parallel processing tools introduced by HEVC, namely WPP and tile processing, for running



(a)



(b)

Fig. 11: The speedup of the CABAC decoding throughput by increasing the decoding parallelism using high-level parallel processing tools in HEVC. (a) and (b) shows the results from WPP and tile processing, respectively. The bitstreams used are the common sequences as listed in Table IV. For each testing bitstream, its decoding throughput without enabling WPP or tile processing is used as the baseline (speedup of $1\times$).

	Avg. Cycles/CTU	Std. Dev. Cycles/CTU
***Nebuta* AI, QP=22**	7376.5	1808.0
***BQ Terrace* LD, QP=37**	59.1	98.1

TABLE VIII: The average and standard deviation of required decoding cycles per CTU.

multiple CABACs in parallel. The design features a deeply pipelined structure and reduces stalls using techniques such as the state prefetch logic, latch-based context memory and separate FSMs. It is also capable of decoding up to two bypass bins per cycle. The benefits of these techniques are summarized in Table IX. The decoder achieves up to 1.06 bin/cycle for high bit-rate common test bitstreams, and 1.44 bin/cycle under the theoretical worst-case scenario. With the clock rate at 1.6 GHz after place-and-route, the throughput reaches 1696 Mbin/s, which is sufficient to real-time decode high-tier video bitstreams at level 6.2 (8K UHD at 120 fps). For applications requiring sub-frame latency, it also supports real-time decoding main-tier bitstreams at level 5.1 (4K UHD at 60 fps).

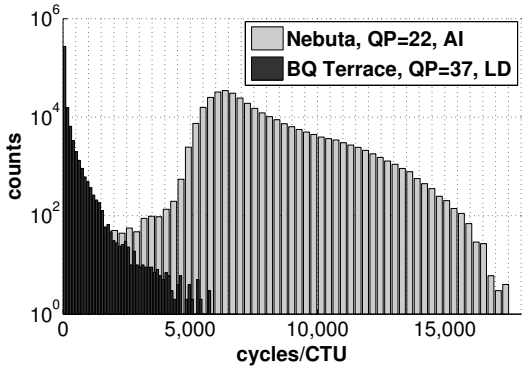


Fig. 12: Histogram of the required decoding cycles per CTU for two cases of bitstreams: AI-coded *Nebuta* at QP=22 and LD-coded *BQ Terrace* at QP=37. The former has high bit-rate and high throughput, and the latter has low bit-rate and low throughput as listed in Table IV.

Technique Applied	Benefit
Deeply Pipelined Architecture (Section II-B and II-C)	high clock rate at 1.6 GHz after place and route
State Prefetch Logic (Section II-B)	reduces the impact of stalls to only 12% throughput degradation without affecting the critical path[1]
Latch-based Memory (Section II-E and IV-A)	reduces overall area and power by 17.3% and 52.0%, respectively (compared to register-based design)
Separate FSM for bypass bins (Section III-A)	increases throughput by up to 33%[1]
Multi-bypass-bin Decoding (Section III-B)	increases throughput by up to 15%[1]

[1] with test sequence *Nebuta* (QP = 22)

TABLE IX: Summary of the proposed techniques

APPENDIX A
WORST-CASE PERFORMANCE ANALYSIS

According to [1], the worst-case performance at three different granularities: within a CTU, within a frame, and across multiple frames, are derived as follows:

- **Within a CTU**: HEVC defines that the maximum number of coded bits for a CTU should be less than

$$\frac{5}{3} * (\mathrm{CtbSizeY} * \mathrm{CtbSizeY} * \mathrm{BitDepth_Y} + 2 * (\mathrm{CtbWidthC} * \mathrm{CtbHeightC}) * \mathrm{BitDepth_C}) \quad (2)$$

where $\mathrm{CtbSizeY}$ is the size of the luma coding tree block (CTB), $\mathrm{CtbWidthC}$ and $\mathrm{CtbHeightC}$ are the width and height of the chroma CTB, respectively, and $\mathrm{BitDepth_Y}$ and $\mathrm{BitDepth_C}$ are the bit depths of luma and chroma samples, respectively. For a given number of coded bits, the maximum number of corresponding bins are achieved when the bit-to-bin ratio is the minimum. For context-coded bins, since the minimum probability of the least probable symbol of CABAC in HEVC is 0.01875 before quantization [3], according to the Shannon entropy theorem, the minimum bit-to-bin ratio is

$$-log_2(1 - 0.01875) = 0.02731. \quad (3)$$

For bypass bins, the ratio is simply 1, and we ignore terminate bins since they take up less than 1% of the total number of bins.

When analyzing the maximum bin-rate under worst-case scenario, the luma CTB size is assumed to be the smallest defined size of 16×16 samples based on the fact that more CTUs in a frame implies more total bins. In this case, $\mathrm{CtbSizeY}$ is 16, $\mathrm{CtbWidthC}$ and $\mathrm{CtbHeightC}$ are 8, and the bit depths $\mathrm{BitDepth_Y}$ and $\mathrm{BitDepth_C}$ are both assumed to be 8. According to Eq. 2, the maximum number of coded bits is 5120 per CTU.

Since the bit-to-bin ratio of the context-coded bins are much lower than that of the bypass bins, the 5120 coded bits are assumed to be composed by the maximum possible number of context-coded bins plus bypass bins for the rest. This is achieved by setting the size of coding blocks, prediction blocks and transform blocks in the CTU to be 8×8, 4×8 (or 8×4) and 4×4 luma samples, respectively. The prediction mode is assumed to be the inter-prediction mode, which signals more bins than the intra-prediction mode. Based on these assumptions, the maximum number of context-coded bins (Table X) is 882. According to Eq. 3, it will be compressed into approximately 24 bits under the minimum bit-to-bin ratio. Therefore, the number of bypass bins is $5120 - 24 = 5096$. By adding up the context-coded and bypass bins, the theoretical maximum number of bins per CTU is 5980.

- **Within a Frame**: The maximum number of bins per frame, $\mathrm{BinCountsInNalUnits}$, is defined to be less than or equal to

$$\frac{32}{3} * \mathrm{NumBytesInVclNalUnits} + \frac{\mathrm{RawMinCuBits} * \mathrm{PicSizeInMinCbsY}}{32}. \quad (4)$$

In this equation, $\mathrm{RawMinCuBits} * \mathrm{PicSizeInMinCbsY}$ can be computed in most common cases as

$$\mathrm{PicWidth_Y} * \mathrm{PicHieght_Y} * \mathrm{BitDepth_Y} + 2 * (\mathrm{PicWidth_C} * \mathrm{PicHieght_C} * \mathrm{BitDepth_C}) \quad (5)$$

where $\mathrm{PicWidth_Y}$ and $\mathrm{PicHeight_Y}$ are the frame width and height in luma samples, respectively, and $\mathrm{PicWidth_C}$ and $\mathrm{PicHeight_C}$ are the frame width and height in chroma samples, respectively. $\mathrm{NumBytesInVclNalUnits}$ is defined under two cases. First, if the frame is the first frame of a sequence, it is defined as

$$\mathrm{NumBytesInVclNalUnits} = \frac{1.5}{\mathrm{MinCR}}\Big(MAX(\mathrm{PicSizeInSamplesY}, \frac{\mathrm{MaxLumaSr}}{300}) + \mathrm{MaxLumaSr} * \mathrm{AuCpbExtraTime}\Big) \quad (6)$$

where MinCR is the minimum compression ratio, $\mathrm{PicSizeInSamplesY}$ is the number of luma samples within a frame, and $\mathrm{MaxLumaSr}$ is the maximum luma sample rate. $\mathrm{AuCpbExtraTime}$ is defined as $\mathrm{AuCpbRemovalTime[0]} - \mathrm{AuNominalRemovalTime[0]}$ in [1], and is the additional time that the first frame will stay in the coded picture buffer (CPB) on top of the nominal duration due to the large frame size. In common cases, it is assumed to be zero. Second, if the frame is not the first frame of a sequence, it is defined as

$$\mathrm{NumBytesInVclNalUnits} = \frac{1.5}{\mathrm{MinCR}} * \mathrm{MaxLumaSr} * \mathrm{AuCpbStayTime} \quad (7)$$

Syntax Element (SE)	Context-coded Bins per SE	Num. SE per CTU
sao_merge_left_flag	1	1
sao_merge_up_flag	1	1
sao_type_idx_luma	1	1
sao_type_idx_chroma	1	1
split_cu_flag	1	1
cu_transquant_bypass_flag	1	4
cu_skip_flag	1	5
pred_mode_flag	1	4
part_mode	3	4
merge_flag	1	8
inter_pred_idc	2	8
ref_idx_l0 (or l1)	2	8
mvp_l0_flag (or l1)	1	8
abs_mvd_greater0_flag	1	8
abs_mvd_greater1_flag	1	8
rqt_root_cbf	1	4
split_transform_flag	1	4
cbf_cb	1	4
cbf_cr	1	4
cbf_luma	1	16
cu_qp_delta_abs	5	4
transform_skip_flag	1	16
last_sig_coeff_x_prefix	3	24
last_sig_coeff_y_prefix	3	24
sig_coeff_flag	1	360
coeff_abs_level_greater1_flag	1	192
coeff_abs_level_greater2_flag	1	24

TABLE X: The distribution of context-coded bins within a CTU under the maximum context-coded bins assumption. The sizes of a CTB, CB, PB and TB are 16×16, 8×8, 4×8 (or 8×4), and 4×4, respectively.

where $\mathrm{AuCpbStayTime}$ is defined as $\mathrm{AuCpbRemovalTime[n]} - \mathrm{AuCpbRemovalTime[n-1]}$ for the n^{th} frame in [1], and is the duration of time that the frame would stay in CPB. This time is usually assumed to be the reciprocal of the sequence frame rate. In most common cases, Eq. 6 and 7 can be computed as

$$\mathrm{NumBytesInVclNalUnits} = \frac{1.5}{\mathrm{MinCR}} * \frac{\mathrm{MaxLumaSr}}{\mathrm{FR}}$$

where FR is the frame rate.

- **Across Multiple Frames**: HEVC directly defines the maximum bit-rate, MaxBR, at each bitstream level, and also restricts the maximum overall bin-to-bit ratio as $4/3$ using *cabac_zero_word*. Therefore, the maximum bin-rate across multiple frames is $4 * \mathrm{MaxBR}/3$ for frames within CPB.

REFERENCES

[1] *High efficiency video coding*. ITU-T Recommendation H.265 and ISO/IEC 23008-2, April 2013.

[2] G. J. Sullivan, J. Ohm, T. K. Tan, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 22, no. 12, pp. 1649–1668, December 2012.

[3] D. Marpe, H. Schwarz, and T. Wiegand, "Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 13, no. 7, pp. 620–636, July 2003.

[4] D. Zhou, J. Zhou, W. Fei, and S. Goto, "Ultra-high-throughput VLSI Architecture of H.265/HEVC CABAC Encoder for UHDTV Applications," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. PP, no. 99, pp. 1–11, July 2014.

[5] V. Sze and M. Budagavi, "High Throughput CABAC Entropy Coding in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 22, no. 12, pp. 1778–1791, December 2012.

[6] Y. Yi and I.-C. Park, "High-Speed H.264/AVC CABAC Decoding," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 17, no. 4, pp. 490–494, April 2007.

[7] Y.-T. Chang, "A Novel Pipeline Architecture for H.264/AVC CABAC Decoder," in *Proceedings of IEEE Asia Pacific Conference on Circuits and Systems*, November 2008, pp. 308–311.

[8] P.-C. Lin, T.-D. Chuang, and L.-G. Chen, "A Branch Selection Multi-symbol High Throughput CABAC Decoder Architecture for H.264/AVC," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009, pp. 365–368.

[9] Y.-H. Liao, G.-L. Li, and T.-S. Chang, "A Highly Efficient VLSI Architecture for H.264/AVC Level 5.1 CABAC Decoder," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 22, no. 2, pp. 272–281, February 2012.

[10] K. Watanabe, G. Fujita, T. Homemoto, and R. Hashimoto, "A High-speed H.264/AVC CABAC Decoder for 4K Video Utilizing Residual Data Accelerator," in *Proceedings of Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI)*, March 2012, pp. 6–10.

[11] J.-W. Chen and Y.-L. Lin, "A High-performance Hardwired CABAC Decoder for Ultra-high Resolution Video," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1614–1622, August 2009.

[12] Y. Choi and J. Choi, "High-throughput CABAC codec architecture for HEVC," *Electronics Letters*, vol. 49, no. 18, pp. 1145–1147, August 2013.

[13] Y.-C. Yang and J.-I. Guo, "High-Throughput H.264/AVC High-Profile CABAC Decoder for HDTV Applications," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 19, no. 9, pp. 1395–1399, September 2009.

[14] P. Zhang, D. Xie, and W. Gao, "Variable-Bin-Rate CABAC Engine for H.264/AVC High Definition Real-Time Decoding," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 17, no. 3, pp. 417–426, March 2009.

[15] M.-Y. Kuo, Y. Li, and C.-Y. Lee, "An Area-efficient High-accuracy Prediction-based CABAC Decoder Architecture for H.264/AVC," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2011, pp. 15–18.

[16] V. Sze and A. P. Chandrakasan, "A Highly Parallel and Scalable CABAC Decoder for Next Generation Video Coding," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 47, no. 1, pp. 8–22, January 2012.

[17] J. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards Including High Efficiency Video Coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 22, no. 12, pp. 1669–1684, December 2012.

[18] Y.-H. Chen and V. Sze, "A 2014 MBin/s Deeply Pipelined CABAC Decoder for HEVC," to appear in *IEEE International Conference on Image Processing (ICIP)*, 2014.

[19] C.-H. Kim and I.-C. Park, "High Speed Decoding of Context-based Adaptive Binary Arithmetic Codes Using Most Probable Symbol Prediction," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2006, pp. 1707–1710.

[20] Y. Hong, P. Liu, H. Zhang, Z. You, D. Zhou, and S. Goto, "A 360Mbin/s CABAC Decoder for H.264/AVC Level 5.1 Applications," in *Proceedings of IEEE International SoC Design Conference (ISOCC)*, November 2009, pp. 71–74.

[21] Y. Choi, private communication, April 2014.

[22] Y. Wang, G. Sullivan, and B. Bross, "JCTVC-P1003: High efficiency video coding (HEVC) Defect Report draft 3," Joint Collaborative Team on Video Coding (JCT-VC), January 2014.

Yu-Hsin Chen (S'11) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2009, and the S.M. degree in electrical engineering and computer science from Massachusetts Institute of Technology, Cambridge, MA, USA, in 2013, where he is currently working toward the Ph.D. degree. His research focuses on energy-efficient algorithm, architecture, and VLSI design for computer vision and video coding systems.

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. degree in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 2006 and 2010 respectively. In 2011, she received the Jin-Au Kong Outstanding Doctoral Thesis Prize in Electrical Engineering at MIT.

She has been an Assistant Professor at MIT in the Electrical Engineering and Computer Science Department since August 2013. Her research interests include energy-aware signal processing algorithms, and low-power circuit and system design for portable multimedia applications. Prior to joining MIT, she was a Member of Technical Staff in the Systems and Applications R&D Center at Texas Instruments (TI), Dallas, TX, where she designed low-power algorithms and architectures for video coding. She also represented TI at the international JCT-VC standardization body developing HEVC. Within the committee, she was the primary coordinator of the core experiment on coefficient scanning and coding, and has chaired/vice-chaired several ad hoc groups on entropy coding. She is a co-editor of *High Efficiency Video Coding (HEVC): Algorithms and Architectures* (Springer, 2014).

Prof. Sze is a recipient of the 2014 DARPA Young Faculty Award, 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 Women's Fellowship for Leadership in Microelectronics in 2008.