Efficient Binary Arithmetic Encoder for HEVC with Multiple Bypass Bin Processing

Quang-Linh Nguyen, Dinh-Lam Tran, Duy-Hieu Bui, Duc-Tho Mai, Xuan-Tu Tran
SISLAB, VNU University of Engineering and Technology (VNU-UET),
144 Xuan Thuy road, Cau Giay district, Hanoi, Vietnam
Email: nqlinh.95@gmail.com, tutx@vnu.edu.vn

***Abstract* — The increasing amount of digital video with supreme quality requires more efficient compression. As the complexity of video coding algorithm is rising, there are more demands for hardware accelerators and customized hardware. Context-based Adaptive Binary Arithmetic Coding (CABAC) is the only entropy coding method adopted in the latest video compression standard, High Efficiency Video Coding (HEVC). Binary Arithmetic Encoder (BAE) is an essential component in CABAC, where the compression process happens. Because of the high data dependency and sequential coding characteristic, it is challenging to parallelize BAE. In this work, we proposed a low-cost and high-throughput hardware architecture for one core of BAE in HEVC. Our 4-stage pipelined BAE architecture is capable of processing one regular bin and up to 4 bypass bins per clock cycle with 30% reduction in terms of area when compared with the designs for one-core CABAC architecture. The design can compress an average of 1.4 bins per cycle. It achieves a throughput of 1 Gbin/s at the maximum operating frequency of 810 MHz with the area of 2.2 kGEs and the power consumption of 2.0 mW in Nangate 45nm technology.**

Keywords* — *HEVC, CABAC, entropy coding, Binary Arithmetic Encoder

I. INTRODUCTION

Digital video has occupied a large share of digital content. In a recent report [1], Cisco forecasted that online video would be responsible for four-fifths of global Internet traffic. This drives the academia and industries to raise the effectiveness of video compression. In this effort, High Efficiency Video Coding (HEVC), introduced by Joint Collaborative Team on Video Coding (JCT-VC), promises 50% bit rate savings compared to the previous standard of H.264/AVC for the same video quality. The standard particularly focuses on two key issues: higher video resolution and increased use of parallel processing architectures [2].

Context-based Adaptive Binary Arithmetic Coding (CABAC) [3] is responsible for entropy coding in HEVC. Each type of its input accompanies with contexts and these contexts need to be updated frequently to adapt to the coding process. CABAC is the slowest part in HEVC because it contains strong data dependency and serial bit processing. Parallelism has been explored to improve the throughput of CABAC in HEVC. HEVC introduced parallelism on frames, slices, or waveform levels to realize high speed in the software version. However, in hardware implementation, using several entropy encoders leads to high hardware cost and high memory footprint. Furthermore, Binary Arithmetic Encoder (BAE) is the main bottleneck in CABAC because it contains bit serial processing of binary data. In this paper, we focus on optimizing this critical module to reduce the hardware cost and improving the throughput.

Recently, there is a large number of works focusing on optimizing the performance of BAE in HEVC. Most of works use the 4-stage pipeline architecture to break BAE into multiple steps with local data dependencies. One way to speed up CABAC is to use many pipelined BAE cores to process multiple bins such as in [13], [12] and [15]; however, this leads to high occupied area and long critical path. It is also possible to use 3 custom cores for 3 types of bins as in [11], but this increases gate count due to the lack of hardware sharing.

In this paper, we propose a 4-stage pipelined architecture for one unified core BAE which is able to process all types of bins at high throughput. In addition, our proposed architecture can process a packet of 4 bypass bins in one clock cycle with the same datapath. Data processing order is reorganized to reduce the critical path for both regular bin and bypass bin. To save the hardware area, we merge the processing of multiple bypass bins with the common path of the regular bins. Implementation results show that our architecture can achieve the speed of 1Gbin/s at the working frequency of 810 MHz with the hardware cost of 2.2 KGEs. At this speed, it is possible to process an 8k videos at high profile in HEVC.

The rest of the paper is organized as followed. Section II introduces briefly about the CABAC algorithm. Section III discusses the multiple-bypass-bin processing technique. Section IV gives details about our proposed hardware architecture for BAE. The implementation result is presented in Section V. Finally, Section VI concludes the paper.

II. CABAC OVERVIEW

Context-based Adaptive Binary Arithmetic Coding is a tool for entropy coding first adopted in H.264/AVC and continuously used in the latest standard HEVC [6]. It is utilized at the last step of video encoding when it will encode the outputs of the previous stages such as quantized transform coefficients, prediction modes, motion vectors, intra prediction

direction, which are called syntax elements (SEs). SE describes how the video can be reconstructed at the decoder.

CABAC encoding process includes three main steps: binarization, context modeling, and binary arithmetic encoding as described in Fig. 1. In the first step, syntax elements are mapped to binary symbols (bins). Context modeler provides the estimated probability of bins. Finally, binary arithmetic encoder compresses bins to bits based on the context model using the provided probability.

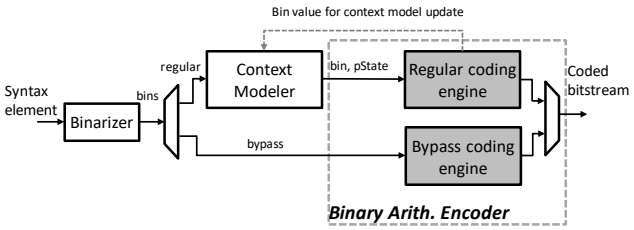

Fig. 1. CABAC encoder block diagram.

Binary arithmetic coding is an extension of arithmetic coding [5] that is used for binary data. As the source data contains only two symbols, there is no need of a statistical structure for the data. The frequency of appearance is recorded after a symbol is coded. The symbol with the probability of at least 0.5 is called Most Probable Symbol (MPS) and the other one is Least Probable Symbol (LPS).

The input of BAE is categorized into three types: regular bin, bypass bin, and terminate bin. Each has a different coding process. While encoding process for regular bin is rather standard, there is no need for a probability model when coding bypass bin and context model is non-adaptive in the case of terminate bin.

A. Binary Arithmetic Encoder

Regular bin coding is the main activity of BAE. Therefore, for hardware implementation, we made some rearrangements to the bypass bin and terminate bin encoding process so that they fit in the main process.

The internal state of arithmetic coding is expressed by two parameters: *Range* (the current interval range) and *Low* (the lower bound of this range). The provided context information includes the probability state index *pState* and the value of MPS *valMPS*.

The process has four main steps and its flow chart is presented in Fig. 2. In the first step, the current interval is divided according to the given probability estimation. The interval subdivision is performed in a multiplier-free fashion, as the range of LPS is selected from a look-up table. MPS range is the result of a subtraction of LPS range from *Range*. Then, *Low* and *Range* are updated according to the type of symbol, MPS or LPS. MPS corresponds to the lower part of the interval range part of the interval range while LPS corresponds to the higher one. The update of probability state is performed in the third step. The last step is the renormalization of *Range* and *Low*.

Since there is a limited number of bits to represent *Range* and *Low*, they need to be scaled up to guarantee the precision. Most significant bits (MSBs) of *Low* will be outputted during the renormalization process. Renormalization happens when *Range* is below the threshold value, 256. After each round, a bit can be generated or accumulated. Accumulated outstanding bits will be resolved when a bit is produced next time. The loop will be iterated until *Range* exceeds 256.

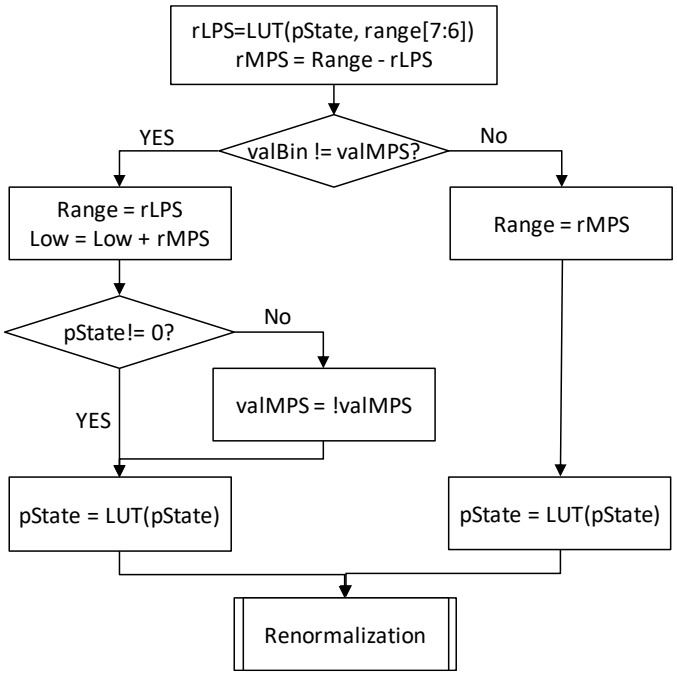

Fig. 2. Flow chart of encoding a regular bin [6].

As context update can be delegated to Context Modeler, the binary arithmetic encoding process can be arranged in four stages according to the order of the updates of *Range*, *Low* and outstanding bit count.

B. State-of-the-art

The trend of designing hardware for CABAC started in 2003 when it was first introduced for H.264 standard by Marpe *et al*. [3]. Most of the works were focus on architecture for Binary Arithmetic Encoder (BAE) because this is the bottleneck of the whole CABAC module.

One of the first to attempt to process multiple bins is Osorio *et al*. in [7]. They implemented 2-stage pipelined BAE with dual-symbol encoding and optimized processing for bypass bin, which resulted in 1.9 – 2.3 bins/cycle. Zheng *et al*. in [8] proposed 4-stage pipelined CABAC with 3-stage pipelined BAE that yielded throughput of 1 bin/cycle. Tian *et al*. in [9] presented three-stage pipelined BAE, one stage for renormalization and two for bit packing. There were three customized submodules used to encode regular bin, bypass bin and terminate bin. Chen *et al*. in [10] designed a dual-core 6-stage pipelined BAE, which gave an average throughput of 2.37 bins/cycle.

As new advanced semiconductor technologies have significantly reduced latency and enabled higher clock rate,

more designs for BAE in HEVC utilized many-core architectures to maximize BAE's throughput. Pham *et al*. in [11] used each custom core to encode each type of bins, which processes 1 bin per cycle. Jo *et al*. in [12] presented 2 parallel 4-stage pipelined BAE cores to boost the performance. They adopted a LUT for generating bitstream to reduce the operational time involved. Zhou's work in [13] is considered as the state-of-the-art architecture for BAE. They implemented 4 parallel pipelined multi-bin BAE cores, which can encode up to 4 regular bins per cycle. The proposed optimizations including bypass bin splitting, pre-normalization, hybrid path coverage, and look-ahead rLPS remarkably reduced the critical path delay of BAE. However, the usage of many BAE cores can lead to high occupied area. In this work, we propose a low-cost solution that balances performance, power and cost.

III. Multiple-Bypass-Bin Processing

For bypass bins, the two bin values, 0 and 1, are equiprobable. Thus, there is no need of a context model to encode bypass bin. Without using probability models, bypass coding engine considerably reduces the coding complexity compared to the regular coding engine. It can be extended to process multiple bins at the same time.

The procedure to encode bypass bins is shown in Fig. 3. The bin with value *binVal* = 1 is assigned to the upper part of the range, *binVal* = 0 to the lower one. Contrary to regular bin coding, in bypass mode, *Low* is first rescaled. Then it is added by the value of *Range* when a '1' is coded, equivalent to the update of *Low* in case of MPS in regular mode. Because the ranges of LPS and MPS are equalized and are half of *Range*, the updated *Range* equals to the old one and renormalization happens for just one round. *Range* is kept unchanged and *Low* is increase by a constant amount of *Range* only when bit '1' is being coded. This leads to the possibility to implement a multiple-bypass-bin coding engine.

As stated in [4], one of the techniques used to improve the throughput of CABAC in HEVC is grouping bypass coded bins. Bins are reorganized in the fashion that bypass bins are grouped together in order to increase the possibility that multiple bins are processed per cycle. Thus, the ability to encoding several bypass bins in a cycle would yield a significant rise in throughput. In addition, bypass bins occupy a noticeable share in the total number of bins, ranging from 20% to 30% of all bins [13]. Our architecture supports encoding of a group of 4 bypass bins in one clock cycle. The group of bypass bins is described using *EPbits* and *EPlen*. *EPbits* is a string of up to 4 bypass bins and *EPlen* indicates the number of bins in *EPbits*.

In bypass mode, renormalized *Low* when coding one bypass bin is computed as follows:

$$renormLow = Range \times binVal + oldLow << 1$$

The first term is *Range* when the bin is '1' and 0 when it is '0'. For the next bypass bin, *Range* remains the same and *Low* is doubled. Therefore, to encode a group of bypass bins, the term is the multiplication of *Range* and *EPbits*. In our design, the formula for multiple-bypass-bin processing is:

$$renormLow = Range \times EPbits + oldLow << EPlen$$

With the ability to encode many bypass bins, our BAE core only takes an additional multiplier compared to the normal core. Furthermore, we also unified the datapath for processing regular bins and bypass bins to calculate the *Low* value.

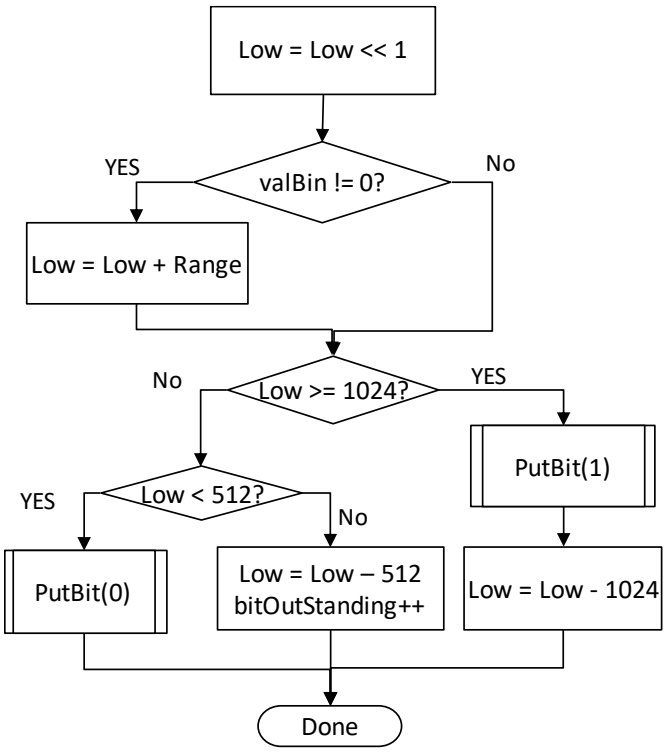

Fig. 3. Flow chart of encoding bypass bin[6].

IV. Proposed Architecture

At Binary Arithmetic Encoder (BAE), bins are encoded serially. Since the probability model and the internal state of arithmetic coding (*Range* and *Low*) should be maintained and updated before coding the next bin, a strong correlation exists between adjacent bins. Therefore, the incoming bin could not be correctly coded until all the necessary computing and updating process for the previous one is completed. This makes BAE rather challenging to parallelize. However, it is possible to break the process down into separate steps which could be pipelined to speed up the coding process. In our proposed architecture, we propose a four-stage pipelined architecture with considerations to maximize the performance and also reduce the occupied area. The area is reduced by passing minimum data through each pipeline stage while the performance is maximized by retiming in different stages. Our proposed architecture also supports to process multiple bypass bins in one clock cycle.

Figure 4 shows our proposed 4-stage pipelined architecture for BAE. To enable multiple-bypass-bin processing, the inputs to our architecture are encapsulated into packets. Each packet could be a regular/terminate bin or a group of bypass bins. The functions of each stage are:

- Stage 1: Packet information extraction and *rLPS* look-up
- Stage 2: *Range* renormalization and pre-multiple bypass bin multiplication.
- Stage 3: *Low* renormalization and outstanding bit look-up
- Stage 4: Coded bit construction and calculation of the number of valid coded bits

The detailed implementation of these stages is described in the following subsections.

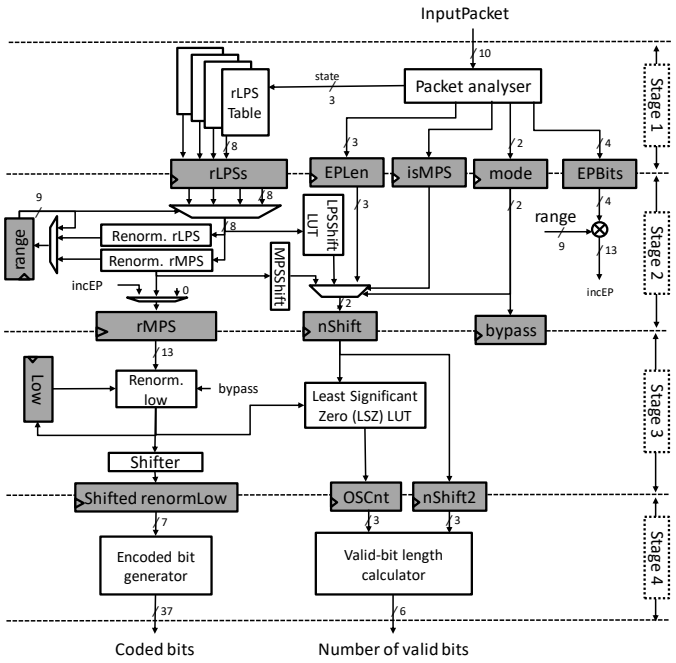

Fig. 4. Our proposed BAE architecture.

A. Packaet information extraction and rLPS look-up

The input of this stage is a 10-bit packet from the context modeler with a specific format for each type of bins. There are 3 types of bins: regular bin, terminate bin, and bypass bin. In our architecture, regular bins and terminate bins have the same data path and they are considered as one type of bins (regular bin). To enable the possibility to process multiple bypass bins, we grouped up to 4 bypass bins together. Therefore, each input packet can be encapsulated as a regular bin or a group of bypass bins as illustrated in Fig. 5. A bypass packet can contain up to 4 bypass bins, while a normal packet carries only one bin. Depending on 2-bit mode, the packet analyzer will extract the corresponding signals to pass through the rest of the coding process.

For regular/terminal bin processing, some previous works such as in [13] and [15] proposed to do *rLPS* and *rLPS* renormalization in this stage. However, to reduce the number of pipelined registers, we chose to pass only control signals and look up four LPS ranges (*rLPS*) in this stage. *rLPS* renormalization and the number of shifted bits will be pushed to the second stage to reduce the total number of registers and to save the combinational logics because at the second stage, we have enough information to decide which *rLPS* will be used for the coding process.

mode		valBin	State						valMPs
9	8	7	6	5	4	3	2	1	0

(a) Packet format for regular/terminate bins

mode		EPBits				EPLen			'0'
9	8	7	6	5	4	3	2	1	0

(b) Packet format for multiple bypass bins

Fig. 5. Packet format for different bin types supported in our architecture.

B. Range renormalization and pre-multiple bypass bin multiplication

Fig. 6 shows the datapath for stage 2. Stage 2 contains two main steps: range renormalization and pre-multiple bypass multiplication for bypass bin processing in stage 3. The input to range normalization process is an *rLPS* value selected from four *rLPSs* look-up in stage 1. This value is selected based on two bits of the previously renormalized range. The area is saved by doing renormalization for only one correct LPS. If renormalization is done in stage 1, it is hard to know which LPS will be used, and all four renormalized LPSs have to be passed through the stage 2. *rLPS* is renormalized by using a shifter with the number of bit shifted is looked-up using the correct *rLPS*. The correct *rLPS* is also used for calculating the *rangeMPS*. The output to stage 3 is the value of *rMPS* which is the pre-multiplication results (*incEP*) for a bypass packet; zero for a most probable symbol bin; or *rangeMPS* in case of a least probable symbol bin. The range register is updated with the renormalized value of MPS or LPS for regular bins, while it keeps its value in case of bypass bins.

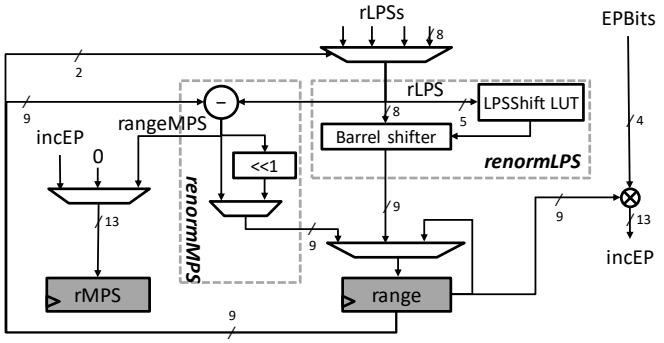

Fig. 6. Range renormalization and pre-multiple bypass bin multiplication architecture.

In our proposed architecture, in this stage, we chose to unify the datapath for bypass bins and regular bins so that the next stage does not need to have two different datapath for two different types of bins. The pre-multiplied value for bypass bin processing is sent to stage 3 through the *rMPS*. In stage 3, the normalization of *Low* value is done using a single adder.

C. Low rernormalization and oustanding bit look-up

Low renormalization and outstanding bit look-up are done in stage 3. In this stage, the area is saved by unifying the datapath for bypass bins and regular bin; sending only 7-bit of the renormalized low which are necessary for the coded bit construction; and by preparing the outstanding bit value in stage 3. Instead of sending all 17 bits of renormalized *Low* value to the next stage, we only have to send 7-bits of then shifted *Low* value and 3 bits of outstanding bit counter (*OSCnt*).

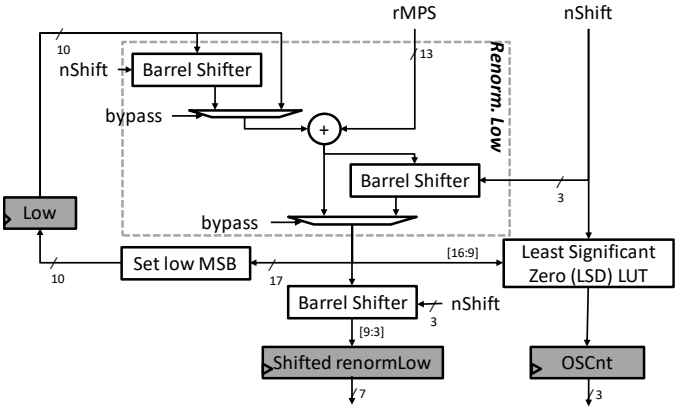

Fig. 7. Low rernormalization and oustanding bit look-up.

The datapath for stage 3 is presented in Fig. 7. In the bypass bin processing, *Low* value is shifted before being added with *rMPS*, while in regular bin processing, the results of the addition of *Low* value and *rMPS* is shifted to form the renormalized *Low*. The *Low* register is updated with 10 bit out of 17-bit of renormalized *Low*. 7 upper bits of renormalized *Low* are used to decide the most significant bit (MSB) of *Low*.

In conventional BAE architecture, the outstanding bit counter is often calculated in the last stage, however, in our architecture, to send only 7-bit of the renormalized low to the next stage, we chose to look up the outstanding bit counter in advance. This path is the critical path in our design.

D. Coded bit construction and calculation of the number of valid coded bits

The last stage in our proposed architecture is shown in Fig. 8. Based on the normalized *Low* value, the coded bits are constructed using inversion, mask with padding zero using barrel shifters. The first bit of renormalized *Low* value is concatenated with its inversion to form a 38-bit value. It is then ANDed with a mask to keep only the resolved outstanding bits. Finally, the resolved bits are ORed with the remaining bits constructed from the last 6 bits of the renormalized *Low* value.

The number of resolved outstanding bits is calculated based on the number of confirmed output bits *bitCnt* and the number of the resolved outstanding bits. If there are no determined output bits, no outstanding bits are resolved and all of them are accumulated into *AccOSCnt* register. Otherwise, outstanding bits from the previous cycle are confirmed and the number of accumulated outstanding bits equals to *OSCnt*.

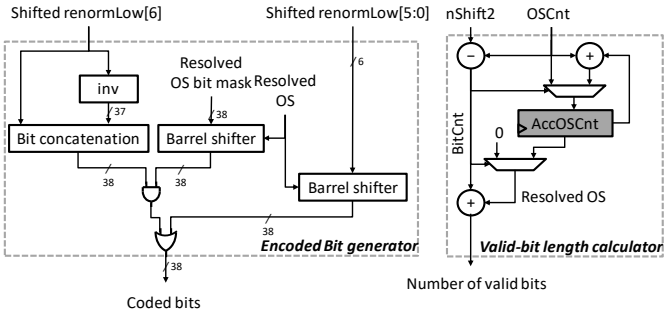

Fig. 8. Stage 4 (Bitstream Generator) architecture.

V. Implementation Result

Our hardware design was implemented in VHDL. We tested our BAE architecture under several test cases. All-intra or low-delay configurations with the quantization parameter of 22 and 37 were configured in the reference software HEVC Test Model version 16.12 as described in TABLE I. With the ability to process at most 4 bypass bins in a clock cycle, our design achieved the performance of 1.4 bins per cycle in average depending on the number of grouped bypass bins. Our proposed architecture was synthesized with Synopsys Design Compiler using NANGATE 45 nm technology. Our architecture has the total gate count of 2.2 kGEs at 810 MHz with power dissipation of 2.0 mW.

TABLE I. Performance Under Different Test Cases

Configuration	QP	Bins per cycle
All intra	22	1.56
	37	1.25
Low delay	22	1.56
	37	1.24
Average		1.4

A comparison of our work with others' works is shown in TABLE II. It is clear from this table that our work achieves the smallest occupied area when compared with the other designs with one-core BAE architecture in [11], [12], [16]. Our proposed architecture has about 30% reduction in area even with multiple bypass bin processing. Furthermore, our design has 20% less power consumption in comparison with 4-core BAE architecture designed for low power in [15] at the same throughput. In terms of throughput, our design at its maximum working frequency achieves 1Gbin/s. This means that our design is capable of encoding in real-time a video conforming to the main profile level 6.2 of high tier [6], which equivalent to an 8K video.

VI. Conclusion

Binary Arithmetic Encoder is the most crucial component of CABAC because of its serial bit-based processing and its internal data dependencies. In this work, we proposed a low-cost and high throughput 4-stage pipelined Binary Arithmetic

Encoder for HEVC CABAC using one-core architecture with multiple-bypass-bin processing. Our architecture can process one regular bin or up to 4 bypass bins in one clock cycle. Our proposed design was successfully implemented using Nangate 45nm technology library with the maximum operating frequency of 810MHz. At this frequency, our design can process 1 Gbins/s, which met the requirement of real-time processing of 8K resolution video. By reorganizing different steps in the pipeline and by retiming technique, our architecture can save 30% of area in comparison with other one-core architectures; and 20% improvement in power consumption even when compared with 4-core architectures designed for low-power consumption.

REFERENCES

[1] Cisco, "White paper: Cisco VNI Forecast and Methodology, 2015-2020," 2016.

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, no. 12, 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,* 2003.

[4] V. Sze and M. Budagavi, "High Throughput CABAC Entropy Coding in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, no. 12, 2012.

[5] G. G. Langdon, "An introduction to arithmetic coding," *IBM Jounal of Research and Development,* 1984.

[6] ITU-T, Recommendation ITU-T H.265 High efficiency video coding, ITU, 2013.

[7] R. R. Osorio and J. D. Bruguera, "High-Throughput Architecture for H.264/AVC CABAC Compression System," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 16, no. 11, 2006.

[8] W. Zheng, D.-X. Li, B. Shi, H.-S. Le and M. Zhang, "Efficient Pipelined CABAC Encoding Architecture," *IEEE Transactions on Consumer Electronics,* vol. 54, no. 2, 2008.

[9] X. Tian, T. M. Le and Y. Lian, Entropy Coders of the H.264/AVC Standard: Algorithms and VLSI Architectures, Springer, 2011.

[10] J.-W. Chen, L.-C. Wu, P.-S. Liu and Y.-L. Lin, "A High-throughput Fully Hardwired CABAC Encoder for QFHD H.264/AVC Main Profile Video," *IEEE Transactions on Consumer Electronics,* vol. 56, no. 4, 2010.

[11] D. H. Pham, J. Moon, D. Kim and S. Le, "Hardware Implementation of HEVC CABAC Binary Arithmetic Encoder," *Journal of IKEEE,* vol. 18, no. 4, 2014.

[12] H. Jo, G. D. A.N and K. Ryoo, "Hardware Architecture of CABAC Binary Arithmetic Encoder for HEVC Encoder," *Advanced Science and Technology Letters,* vol. 141, 2016.

[13] 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,* 2015.

[14] D. Marpe, G. Marten and H. L. Cycon, "A Fast Renormalization Technique for H.264/MPEG4-AVC Arithmetic Coding," *14th European Signal Processing Conference,* 2006.

[15] F. L. L. Ramos, J. Goebel, B. Zatt, M. Porto and S. Bampi, "Low-Power Hardware Design for the HEVC Binary Arithmetic Encoder Targeting 8K Videos," *Symposium on Integrated Circuits and Systems Design (SBCCI),* 2016.

[16] B. Peng, D. Ding, X. Zhu and L. Yu, "A Hardware CABAC Encoder for HEVC," in *IEEE International Symposium on Circuits and Systems*, 2013.

TABLE II. COMPARISON WITH OTHER WORKS

	Peng2013 [16]	Zhou2014 [13]	Pham2014 [11]	Jo2016 [12]		Ramos2016 [15]	**Our work**
Bins per cycle	1.18	**4.37**	1	1	1.99	4	1.4
Clock frequency (MHz)	357	420	180	**1530**	1110	280	810
Technology process (nm)	130	90	180	65	65	45	45
Throughput (Mbin/s)	421	1836	180	1530	**2219**	1120	1134
Gate count (K gates)	24.95	7.98	3.96	3.17	5.68	9.95	**2.2**
Power consumption (mW)	-	-	2.88	-	-	2.49	**2.0**