VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Nguyen Quang Linh

DESIGN AND IMPLEMENTATION OF A LOW-COST, LOW-POWER BINARY ARITHMETIC ENCODER FOR HEVC CABAC

Major: Electronics and Communication Engineering

HA NOI - 2017

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Nguyen Quang Linh

DESIGN AND IMPLEMENTATION OF A LOW-COST, LOW-POWER BINARY ARITHMETIC ENCODER FOR HEVC CABAC

Major: **Electronics and Communication Engineering**

Supervisor: **Assoc. Prof. Tran Xuan Tu**

HA NOI - 2017

AUTHORSHIP

“I hereby declare that the work contained in this thesis is of my own and has not been previously submitted for a degree or diploma at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no materials previously published or written by another person except where due reference or acknowledgment is made.”

Signature:……………………………………………

SUPERVISOR’S APPROVAL

“I hereby approve that the thesis in its current form is ready for committee examination as a requirement for the Bachelor of Electronics and Communication Engineering degree at the University of Engineering and Technology.”

Signature:………………………………………………

ACKNOWLEDGEMENT

First and foremost, I would like to express my sincere gratitude to my supervisor, Associate Professor Tran Xuan Tu, for his patience and belief in me, along with his guiding and support. He gave me the opportunity to work on this interesting topic. I have learned a lot from him, not just about technical subjects but also about the industry, society, and life.

I would also like to thank Dr. Nguyen Kiem Hung, who offered me several insights, my classmates and my colleagues at the VNU Key Laboratory for Smart Integrated Systems (SISLAB), where I carried out this project.

I greatly appreciate Faculty of Electronics and Telecommunications and University of Engineering and Technology for providing me with a great academic environment, where I have learned, worked and grown up every day.

Last but not least, I am grateful to my family for their unconditional love, support, and encouragements.

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 is the only entropy coding method adopted in the latest video compression standard, H.265/HEVC. Binary Arithmetic Encoder 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. Pipeline stands out as an effective solution for this problem. In this work, we proposed a low-cost, low-power hardware architecture for Binary Arithmetic Encoder. Our 4-stage pipelined BAE architecture is capable of processing up to 4 bins per cycle clock. The design can compress an average of 1.4 bins per cycle. The design was modeled using VHDL and then implemented with NANGATE 45 nm technology. It achieves a throughput of 828 Mbin/s at the maximum operating frequency of 591.7 MHz. The area cost is 2.8 K gates and the power comsumption is 0.36 mW at 591.7 MHz.

Keywords: HEVC, CABAC, entropy coding, Binary Arithmetic Encoder, pipeline

TABLE OF CONTENTS

List of Figures

List of Tables

ABBREVIATIONS

HEVC	High Efficiency Video Coding
CABAC	Context-based Adaptive Binary Arithmetic Coding
BAE	Binary Arithmetic Encoder
SE	Syntax element
bin	Binary symbol
MPS	Most probable symbol
LPS	Least probable symbol
LUT	Look-up table
rLPS	LPS range
rMPS	MPS range
LSZD	Least significant zero detector
HM	HEVC Test Model
QP	Quantization parameter
RTL	Register transfer level

Chapter 1

INTRODUCTION

1.1. Motivation

Since the Internet and all its following technology emerged, 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. The ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) standardization organizations are two leading groups in producing video coding standard. In partnership, they established the Joint Collaborative Team on Video Coding (JCT-VC). The most successful product of the team is H.264/MPEG-4 AVC, launched in 2003. It is used for a broad range of applications such as television broadcast, video editing, Internet video, and video conferencing. However, the more diverse set of services and the growing of excellent-quality moving picture such as 8K, 4K resolution videos demand a video coding standard that is more advanced than H.264. High Efficiency Video Coding was introduced to substantially address all existing applications of H.264/AVC and to particularly focus on two key issues: higher video resolution and increased use of parallel processing architectures [2].

With an increasing diversity of services, much more tasks require real-time video processing. As the quality of content is rising, there is also exponential growth in computing workload. Thus, in real-time service, such quantity of computation might exceed a general purpose processor’s capability. Consequently, customized hardware was proposed as the solution. Fully-hardwired video encoder/decoder chips can solely compress/decompress videos, while hardware accelerators can be used as a complement to the main processor to tackle some critical tasks and share computing burden.

Context-based Adaptive Binary Arithmetic Coding [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 process. Therefore, it expresses strong data dependency and serial bit nature, which is most clearly presented in Binary Arithmetic Encoder, the key component of CABAC encoder. With such characteristics, a tailored hardware architecture for BAE is highly necessary.

1.2. Contributions and thesis overview

With this work, we promoted a hardware implementation for Binary Arithmetic Encoder of CABAC in the latest video compression standard, H.265/HEVC. The design can be deployed as a part of full hardware-based HEVC encoder chip. It was tested thoroughly to guarantee the right functional behavior when bringing up to a silicon chip.

The main contributions of this thesis include:

- An overall architecture for CABAC encoder was presented. It is used as a guideline to determine the function of each module and how they interact with each other, as BAE is one of them.

- We proposed a 4-stage pipelined architecture for BAE to meet the requirement of a low-cost, low-power design.

- Bypass and terminate bin encoding path were reorganized to merge with regular bin’s, which enables our one-core design to be fit for all type of inputs.

- In order to utilize some improvements in HEVC compression algorithm for our hardware, we added a feature of processing up to 4 bypass bins in one cycle clock.
- We also rearranged and equitably distributed components for each stage to diminish delay in some critical stages.
- Standard test vectors extracted from HEVC reference software was used to verify our hardware.

The rest of this thesis is organized as follows.

Chapter 2 gives an overview of CABAC. It introduces the background theory and the encoding process of CABAC.

In Chapter 3, we examine the detail of Binary Arithmetic Encoder. Some related works are also discussed.

Chapter 4 presents our proposed hardware architecture for BAE. An overall CABAC encoder architecture is introduced and then each module of BAE is clearly described.

The verification strategy and implementation result of the architecture are reviewed in Chapter 5.

Finally, Chapter 6 concludes the thesis and discusses the future work.

Chapter 2

CONTEXT-BASED ADAPTIVE BINARY ARITHMETIC CODING

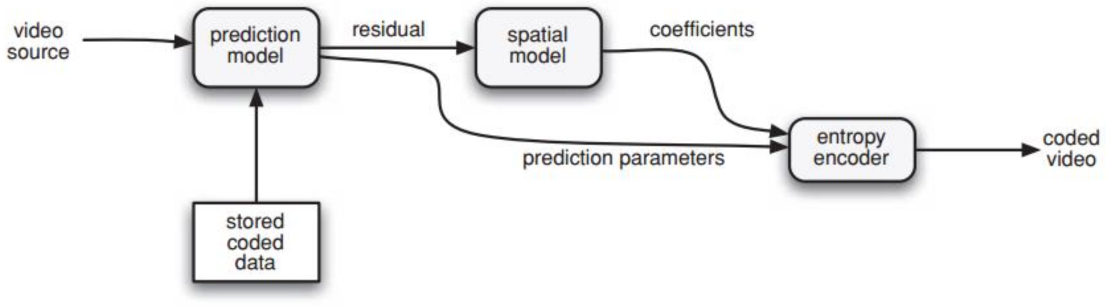


Figure 1: Video encoder block diagram [4].

A video codec compresses a source video sequence into an efficiently coded representation that can be decoded to produce a copy or approximation of the origin sequence. In a regular video encoding system, there are three main functional units: a prediction model, a spatial model and an entropy encoder [4]. The prediction model exploits the similarity between consecutive video frames and outputs residual data, the result of the subtraction of the prediction from the current frame. The residual frame forms the input to the spatial model which is responsible for reducing spatial redundancy. The model transforms residual samples into another domain and quantizes the result. Finally, the prediction parameters and quantization coefficients are compressed by the entropy encoder, which removes statistical redundancy. Arithmetic coding is a well-known lossless entropy coding algorithm.

2.1. Arithmetic Coding

Arithmetic coding is a lossless data compression method [5]. It maps a string of symbol into an interval of real numbers. The initial value of the interval is [0, 1). When each symbol is coded, the current interval is substituted by one of its subinterval, which is selected based on the probability of the currently coded symbol. The decoder can precisely reconstruct the original string if any point in the current interval is provided. The left most point of the interval is normally recorded as the result.

For example, a set of symbols {1, 2, 3} has the corresponding probability {0.8, 0.02, 0.18}. The process of encoding the sequence "1321" is shown as follow.

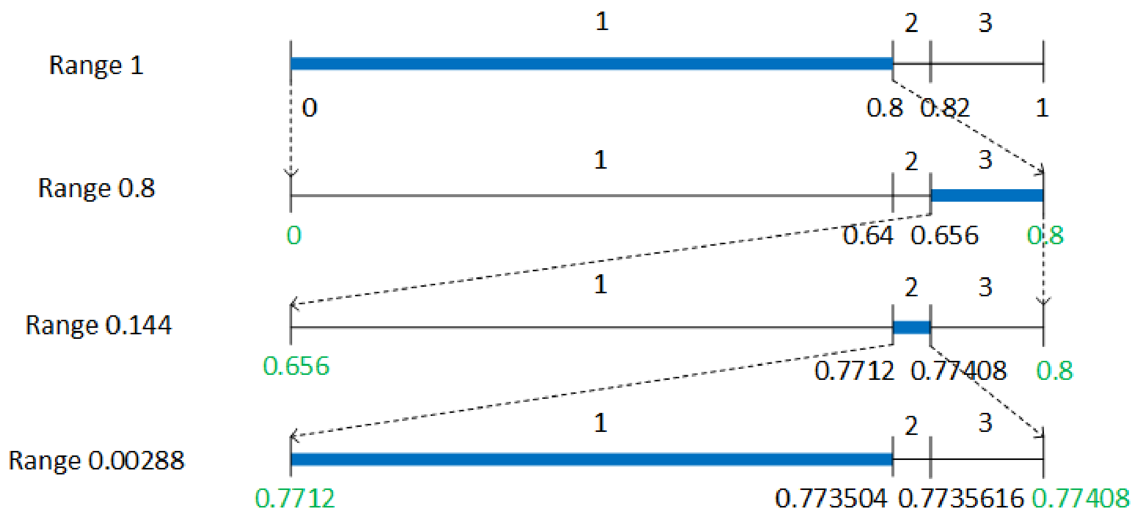


Figure 2: Example of encoding the sequence "1321".

It can be noticed that the algorithm needs high precision to store the interval value and no output is generated until the entire sequence is coded. This is not practical for many applications. Thus, scaling and incremental coding were introduced to deal with these issues. Since the interval range becomes smaller, many most significant bits of two ends of the range become similar. In above example, the binary form of 0.7712 and 0.773504 are 0.**110001**0... and 0.**110001**1.... It can output the identical MSBs and rescale the rest. This allows the algorithm to achieve infinite precision with finite-precision integers.

2.2. CABAC Encoding Process

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].

Entropy coding is a lossless compression technique based on the statistical characteristic of the data source. It is utilized at the last step of video encoding when it will encode the output of previous stages, such as quantized transform coefficients, prediction modes, motion vectors, intra prediction direction, which are called syntax elements. SE describes how the video can be reconstructed at the decoder.

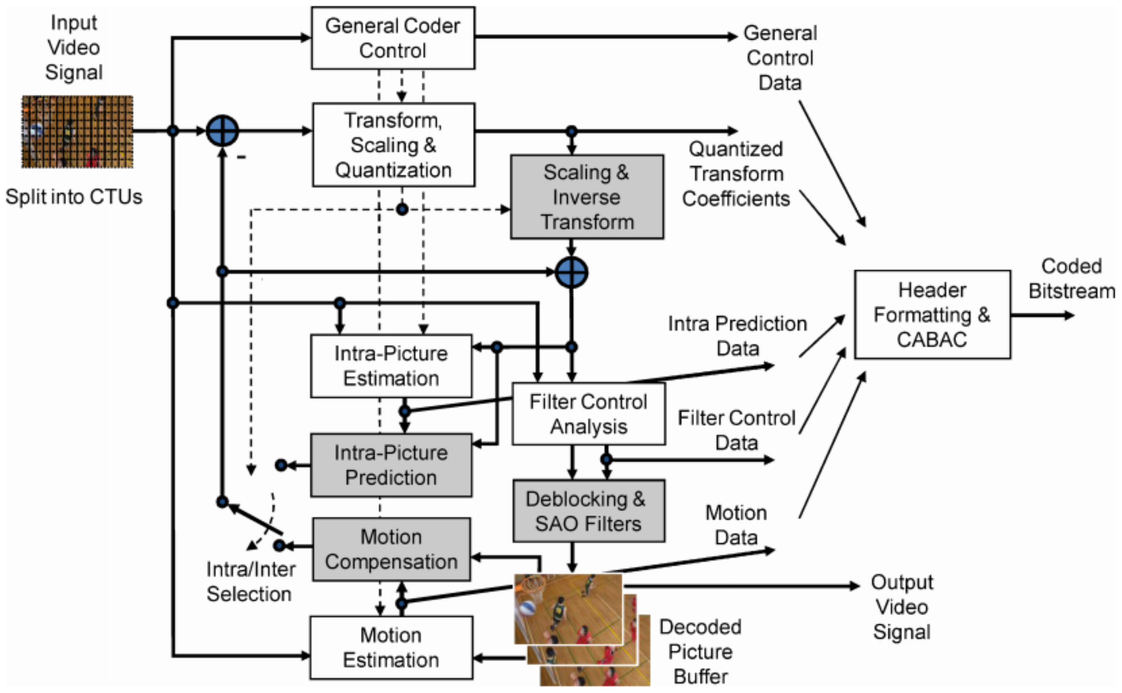


Figure 3: HEVC video encoder block diagram [2].

CABAC encoding process includes three main steps: binarization, context modeling, and binary arithmetic encoding. In the first step, syntax elements are mapped to binary symbols (bins). Context modeling provides the estimated probability of the bins. Finally, binary arithmetic coding compresses bins to bits using the provided probability [6].

We will discuss Binarizer and Context Modeler in the next subsections, and leave the detail of Binary Arithmetic Encoder for the following chapter.

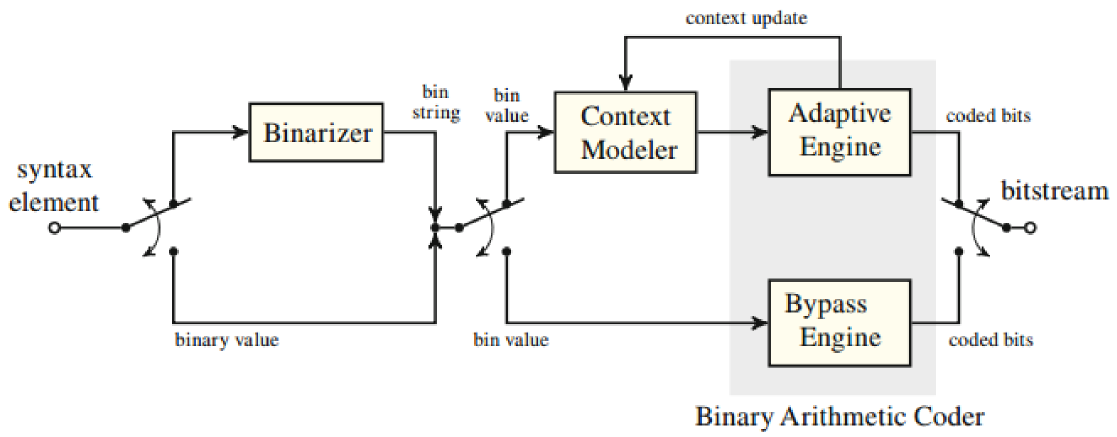


Figure 4: CABAC encoder block diagram [7].

2.2.1. Binarizer

Binarizer performs binarization process. It transforms non-binary syntax elements into a prefix-free string of binary values. Different binarization schemes are used in HEVC, including unary (U), truncated unary (TU), truncated Rice (TRk), kth-order Exp-Golomb (EGK), and fixed length (FL).

Unary coding presents a non-negative integer value v by $v + 1$ bits, which includes v bits of ‘1’ and one last ‘0’ bit.

Truncated unary coding limits the length of binary string by the maximum value that can be presented, *cMax*. The trailing ‘0’ can be omitted if SE value equals to *cMax*.

Truncated Rice, with configuration parameter k, has a unary prefix and k suffix bits. The number of leading ‘1’ bits when encoding a value v is:

$$p = \left\lfloor \frac{v}{2^k} \right\rfloor$$

The suffix is the binary form of *s*.

$$s = v - p.2^k$$

*k*th-order Exp-Golomb code has a unary prefix and a variable-length suffix. The prefix corresponds to the value of:

$$p = \left\lfloor \log_2 \left(1 + \frac{v}{2^k}\right) \right\rfloor$$

The suffix is the binary representation of:

$$s = v + 2^k(1 - 2^p)$$

Table 1: Example of different binarization schemes

N	U	TU cMax = 7	TRk cMax = 7; k = 1	EGk k = 7	FL cMax = 7
0	0	0	00	1	000
1	10	10	01	010	001
2	110	110	100	011	010
3	1110	1110	101	00100	011
4	11110	11110	1100	00101	100
5	111110	111110	1101	00110	101
6	1111110	1111110	1110	00111	110
7	11111110	1111111	1111	0001000	111

Fixed length code has a fixed number of bits, which equals to *ceil(log2(cMax + 1))*. The code corresponds to the binary representation of the value.

Each syntax element is assigned to a binarization scheme based on its type. Some SEs use the combination of mentioned schemes. For some special SEs, custom binarization processes are demanded. Customized schemes may require the value of previous processed SEs or slice parameter which indicate if certain modes are enabled.

2.2.2. Context Modeler

Context modeler has three main functions:

- **Context initialization**: each context is initialized to a probability state *pState* and most probable symbol value. Initialized values of the two variables depend on the 8-bit initialization value and the luma quantization parameter QP_{slice} of the current slice.

- **Context selection**: a context model is assigned to each regular and terminate bin. Context index is used to address each context model in the memory. Selection of context model is based on the type of syntax element, bin position in syntax element (*binIdx*), luma/chroma, and information of previously coded syntax element in the neighborhood, slice type, etc.

- **Context storage and update**: CABAC uses 154 contexts. Each context occupies 7 bits in memory, 6 bits for probability state and 1 bit for most probable symbol value, and is addressed by context index. After a regular bin is coded, the corresponding context model is updated through a table-based state transition.

BINARY ARITHMETIC ENCODER

Chapter 2 provides the function of Binarizer and Context Modeler. This chapter continues with the last and most important component of CABAC, Binary Arithmetic Encoder.

After Binarizer converts SE to a sequence of bins and Context Modeler picks the corresponding probability, BAE applies binary arithmetic coding to each bin.

Binary arithmetic coding is an extension of arithmetic coding 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 and the other one is Least Probable Symbol.

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.

3.1. Regular Bin Encoding

The internal state of arithmetic coding is expressed by two parameters: *Range* (the current interval range) and *Low* (the lower bound of this range) [3]. The provided context information includes the probability state index *pState* and the value of MPS *valMPS*.

The process has four main steps.

In the first step, the current interval is divided according to the given probability estimates. The interval subdivision is performed in a multiplier-free fashion, as the range of LPS is selected from a look-up table based on the value of *pState* and the quantized value of *Range*, the seventh and sixth bit of *Range*. MPS range is the result of a subtraction of LPS range from *Range*.

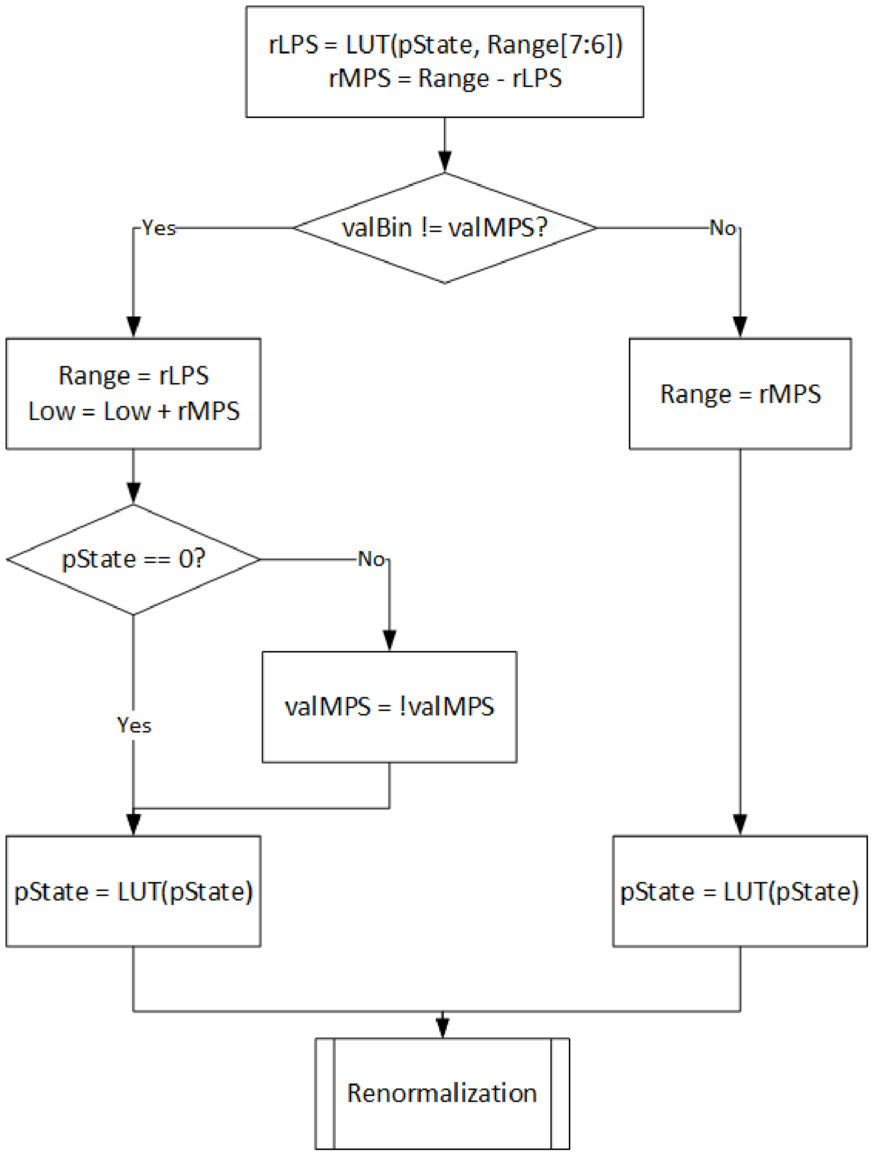


Figure 5: Flow chart of encoding a regular bin [8].

Then, *Low* and *Range* are updated according to the type of symbol, MPS or LPS. MPS corresponds to the lower and LPS to the higher part of the interval range. If the symbol is LPS, *Range* is assigned to LPS range and *Low* increases an amount of MPS range. Otherwise, *Range* is set to MPS range and *Low* remains the same. In this way, *Low* is less likely to be updated.

The update of probability state is performed in the third step.

The last step is the renormalization of *Range* and *Low*.

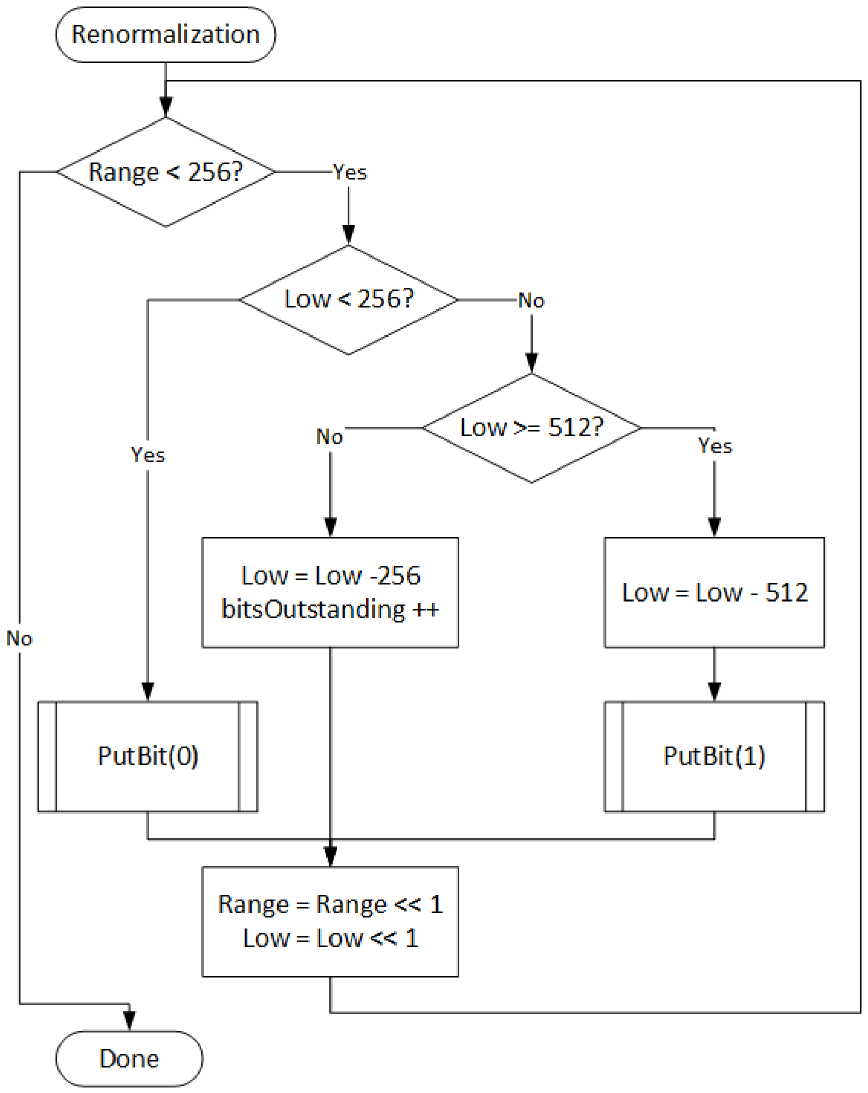


Figure 6: Flow chart of renormalization in the encoder [8].

Renormalization corresponds to rescaling and incremental coding process in arithmetic coding. Since there is a limited number of bits to present *Range* and *Low* (9 bits for *Range* and 10 bits for *Low*), they need to be scaled up to guarantee the precision. MBS of *Low* will be outputted during this process. Renormalization happens when *Range* is below the threshold value, 256. After each round, a bit can be generated, might be followed by a string of inverted bits, or an outstanding bit can be accumulated. Accumulated outstanding bits will be resolved when a bit is produced next time. The loop will be iterated until *Range* exceeds 256.

3.2. Bypass Bin Encoding

For bypass bin, the two bin values, 0 and 1, are equiprobable. Thus, there is no need of context model to encode bypass bin. Without using probability models, bypass engine considerably reduce the complexity compared to the adaptive engine and is capable of processing multiple bins at the same time.

The bin with value *binVal* = 1 is assigned to the upper part of the range, *binVal* = 0 to the bottom. 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 range 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.

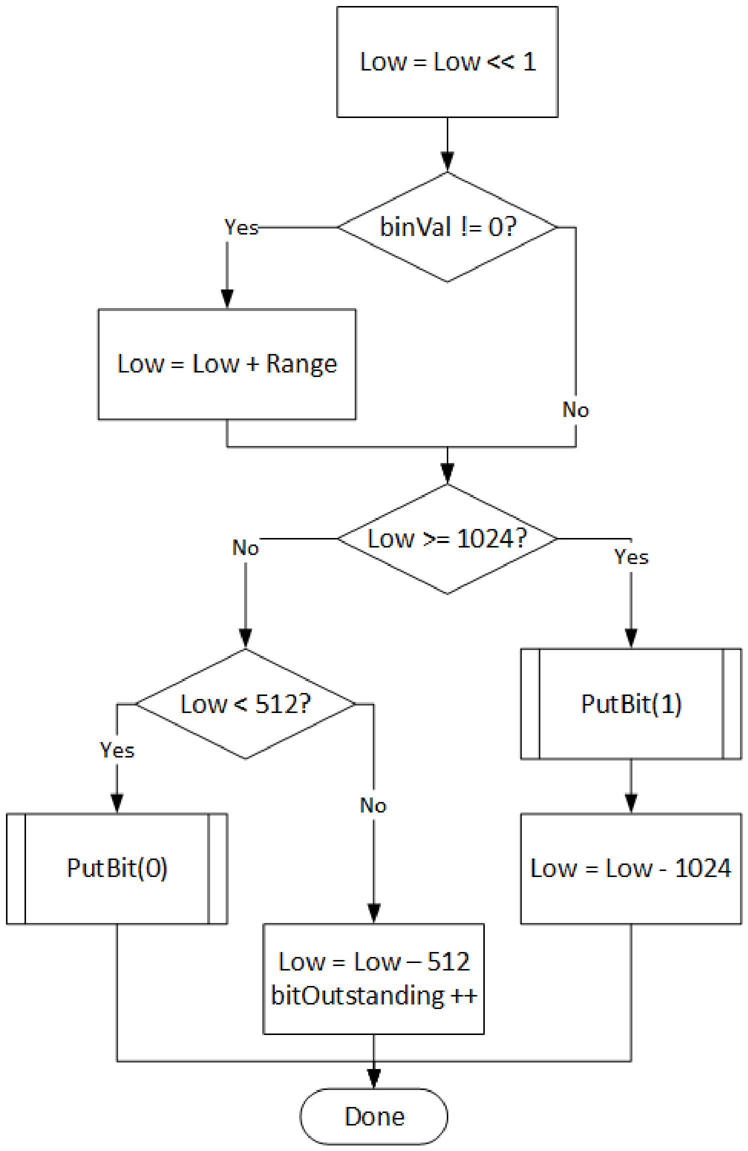


Figure 7: Flow chart of encoding bypass bin [8].

3.3. Terminate Bin Encoding

Terminate bin has a dedicated context model, with context index 0, that associates with non-adaptive probability state 63. LPS is assigned to bin value 1 and has a fixed range of 2. The coding process is quite similar to that of regular bin exception the update of the probability model.

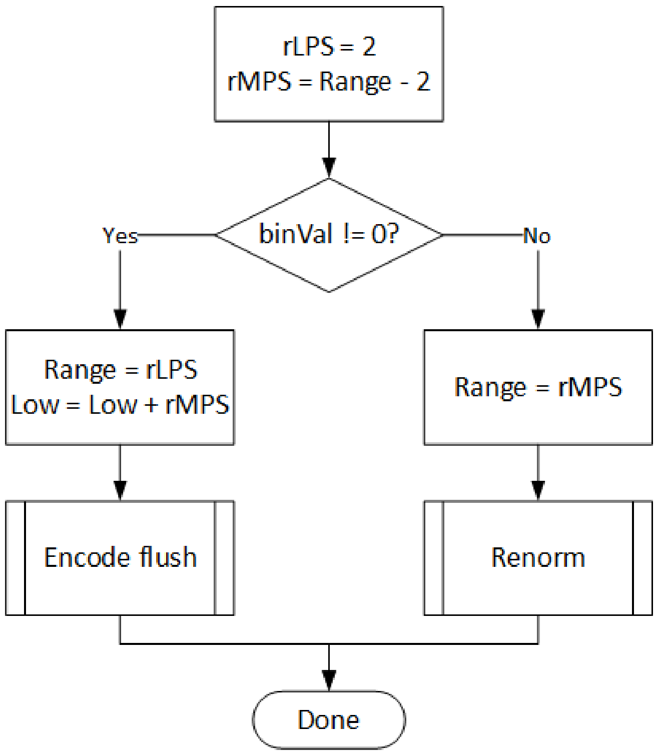


Figure 8: Flow chart of encoding terminate bin [8].

An LPS, in this case, indicates the end of a slice segment, the end of a sub-bitstream or PCM coding. Accordingly, 7 bits are outputted in the renormalization.

3.4. Related Works

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 because this is the bottleneck of the whole CABAC module.

One of the first to attempt to process multiple bins is Osorio *et al*. [9]. They implemented 2-stage pipelined BAE with dual-symbol encoding and optimized processing for bypass bin, which resulted in 1.9 – 2.3 bin/cycle. Zheng *et al*. [10] proposed 4-stage pipelined CABAC with 3-stage pipelined BAE that yielded throughput of 1 bin/cycle. Tian *et al*. [11] presented three-stage pipeline 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*. [12] designed a dual-core 6-stage pipelined BAE with that can encode up to 8 bypass bins at once, which gave the average performance of 2.37 bin/cycle.

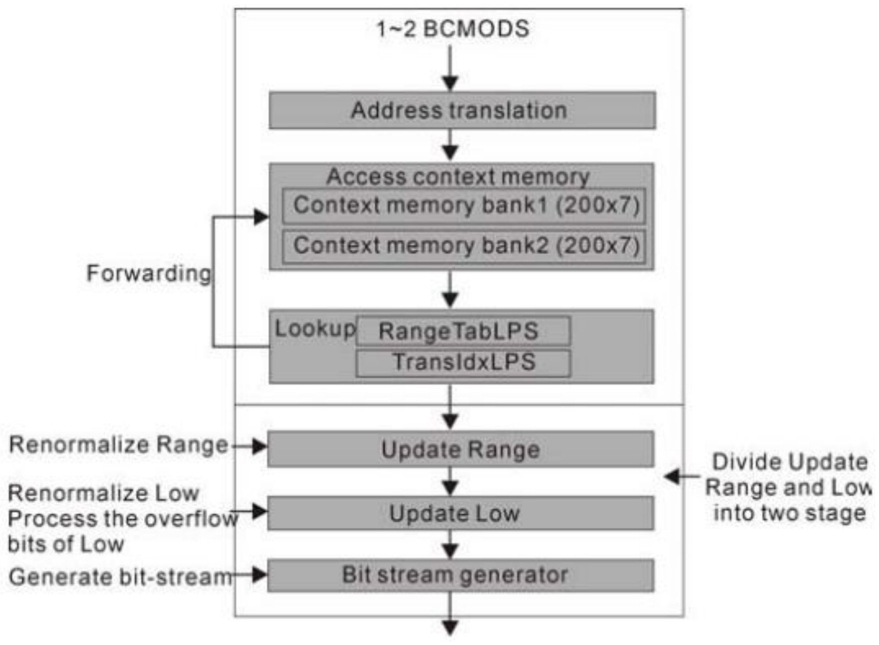


Figure 9: BAE architecture of [12].

Since Sze *et al*. [6] presented CABAC of HEVC in 2012, architectures for BAE have been renovated to adapt to the proposals. As new advanced semiconductor technology have significantly reduced latency and enabled higher clock rate, more works utilized many-core design to maximize the throughput. Some typical works are listed below.

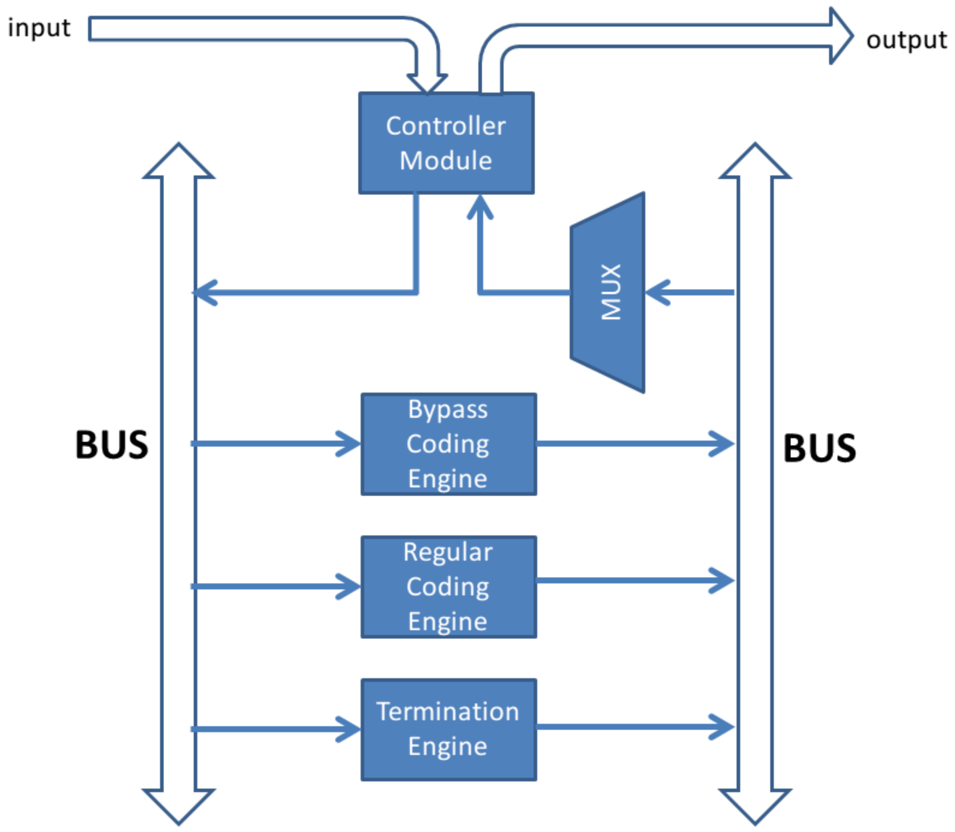


Figure 10: BAE architecture of [13].

Pham *et al*. [13] used each custom core to encode each type of bin, which processes 1 bin per cycle. Jo *et al*. [14] 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 *et al*. [15] s' work is considered the state-of-the-art architecture for Binary Arithmetic Encoder. 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 lookahead rLPS remarkably reduced the critical path delay of BAE.

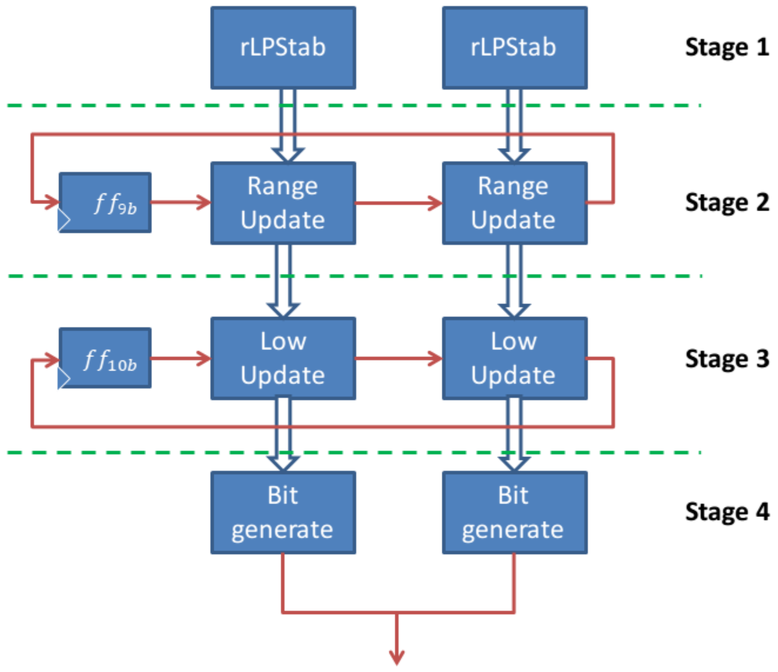


Figure 11: BAE architecture of [14].

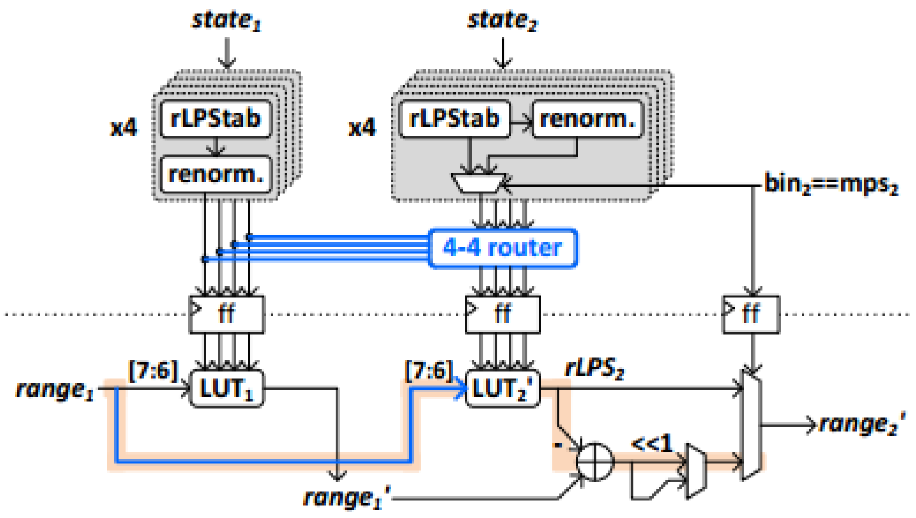


Figure 12: Lookahead rLPS [15].

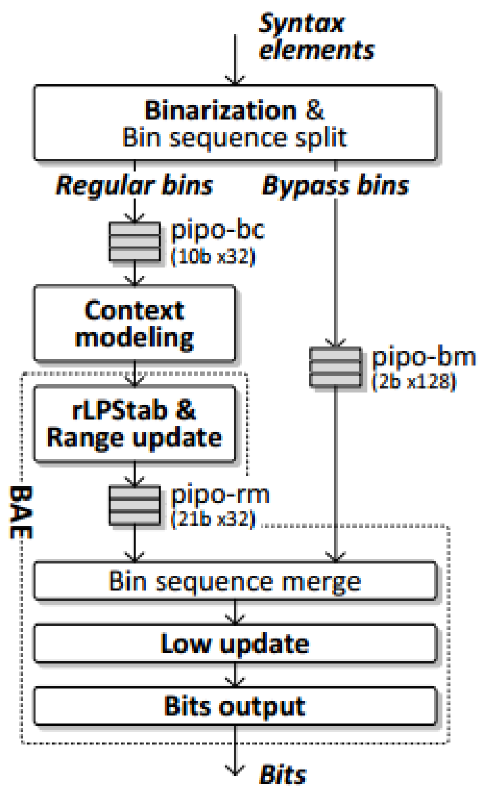


Figure 13: Bypass bin splitting [15].

In summary, processing several regular bins at once requires multi-core architecture, which results in higher area and power cost. Designing each core for each type of input reduces the complexity of the system but still makes use of ample space. Thus, we consider a one-core-fit-all architecture to fulfill the purpose of a low-cost design. Zhou [15] ‘s success was partly attributed to the usage of algorithmic improvements of CABAC in HEVC, which they used bypass bin splitting scheme to process bypass bins in parallel with the pipelined core. Therefore, in the next chapter, we will discuss the benefit of multiple bypass bins processing.

PROPOSED ARCHITECTURE

4.1. CABAC Architecture

As Binary Arithmetic Encoder is a component of CABAC, it is necessary to define the function of each module and the interface between them. Therefore, we designed an overall architecture for CABAC encoder.

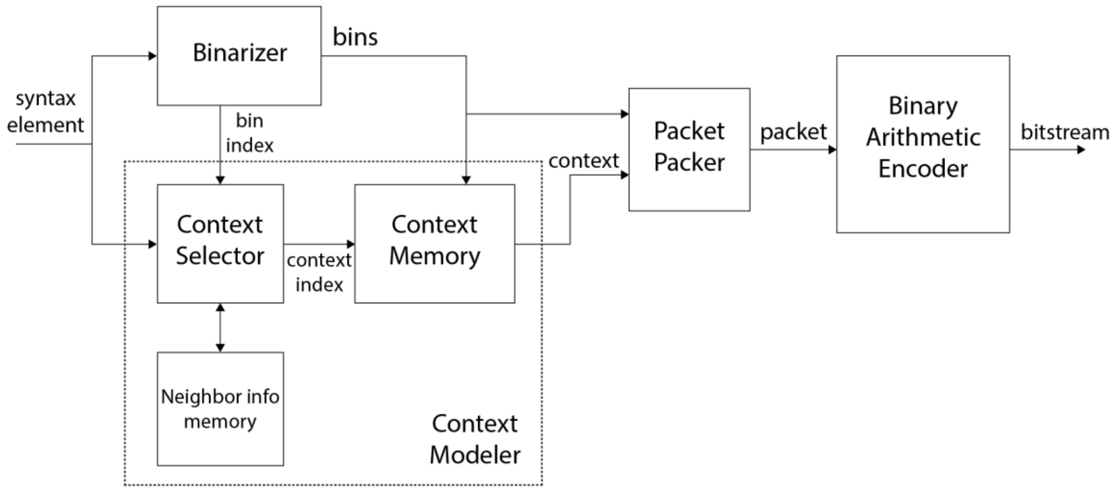


Figure 14: Proposed CABAC encoder architecture.

The expected operation of this architecture is described as follows.

Binarizer translates non-binary syntax elements into strings of binary symbols and assigns a bin index to each bin.

Simultaneously, Context Selector chooses the context model based on the SE type, bin index and/or neighboring SE's information. Its output, context index, is sent to Context Memory as an address for the selected context. Context Memory, with bin value received from Binarizer, also determines whether the bin is MPS or LPS and updates the context accordingly using an LUT.

Bin and the corresponding context information, including the MPS value and the probability state, are coupled up and passed to Packet Packer. Here they are wrapped into packets to fetch to Binary Arithmetic Encoder. A packet can contain one regular bin or one terminate bin or up to 4 bypass bins.

Finally, BAE encodes the packet and generates bitstream.

4.2. Binary Arithmetic Encoder Architecture

At Binary Arithmetic Encoder, 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. These natures make BAE rather challenging to parallelize. However, it is possible to break the process down into steps. Many works have pointed out that there is no feedback among update of each parameter and they can be arranged in each step. Thus, pipeline is the rational solution to optimize BAE.

There are three main trends in designing architecture for BAE. Some works choose to use each custom core for each type of bin. This approach is quite expensive due to lack of hardware shared among cores and usually does not yield a good result, as it only processes 1 bin/cycle and has a fairly long critical path. Other teams use many cores, normally pipelined cores, in parallel to process multiple bins. However, as mentioned above, it is not easy to achieve high performance this way. Due to the mandatory update, it requires several advanced techniques to shorten the forward path between consecutive cores. Thus, only a few works really succeed in applying this method. Whereas, using one pipelined core takes up less area and gives decent performance. Therefore, with an aim for a low-cost, low-power hardware core, a one-core fully pipelined BAE architecture would be reasonable for us.

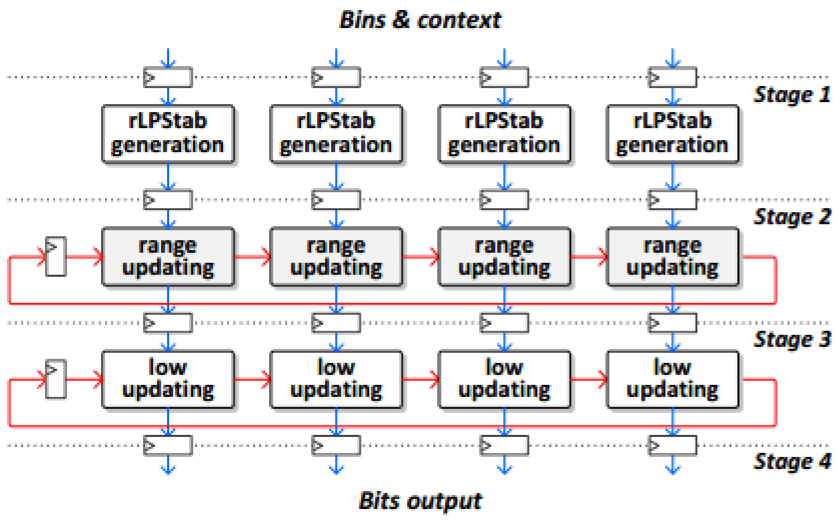


Figure 15: Example of multi-core BAE architecture [15].

As shown in figure 5, BAE's regular bin encoding process contains these steps: interval subdivision, *Range/Low* assignment, probability model update, and *Range/Low* renormalization. Since context update can be delegated to Context Modeler, the rest can be arranged in four stage according to the order of the update of *Range*, *Low* and outstanding bit count. *Range* update happens first since the number of rounds in *Low* renormalization depends on *Range* value. Because the number of new outstanding bits is determined based on the renormalized Low, it accumulates outstanding bits at the end. Thus, the process can be divided into four stages:

- Stage 1: preparation for interval subdivision
- Stage 2: interval division and *Range* renormalization
- Stage 3: *Low* renormalization
- Stage 4: output upper bits of renormalized *Low* and update of outstanding bit count

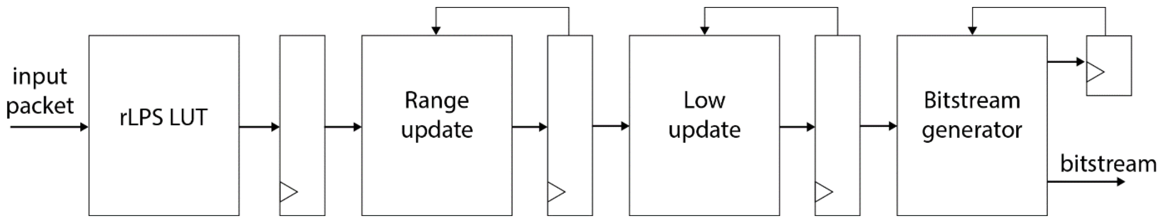


Figure 16: Proposed BAE architecture.

Figure 16 depicts our proposed 4-stage pipelined architecture for BAE. This design can encode all type of bin. While encoding terminate bin is quite similar to coding regular bin, some adjustment is made so that bypass bin encoding fits in the architecture.

One major feature that we implemented in this design is the capability of encoding multiple bypass bins in a clock cycle. In figure 7, it is indicated that in bypass mode, there is no usage of context. *Range* is kept unchanged and *Low* is increase by a constant amount of *Range* only when bit ‘1’ is being coded, which make it effortless to predict. Thus, a string of consecutive bypass bins can be easily processed at one. In addition, bypass bins occupy a noticeable share in the total number of bins, ranging from 20% to 30%.

Table 2: Proportion of bins by type in some sequences encoded by HEVC [15]

Sequence	**MPS (%)**	**LPS (%)**	**Bypass (%)**
BaseketballDrive	54.8	18.1	27.1
Traffic	56.4	21.1	22.5
PeopleOnStreet	50.9	19.9	29.2
BQTerrace	61.6	18.4	20
Kimono	52.7	18.2	29.1

As stated in [6], 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.

4.2.1. rLPS LUT

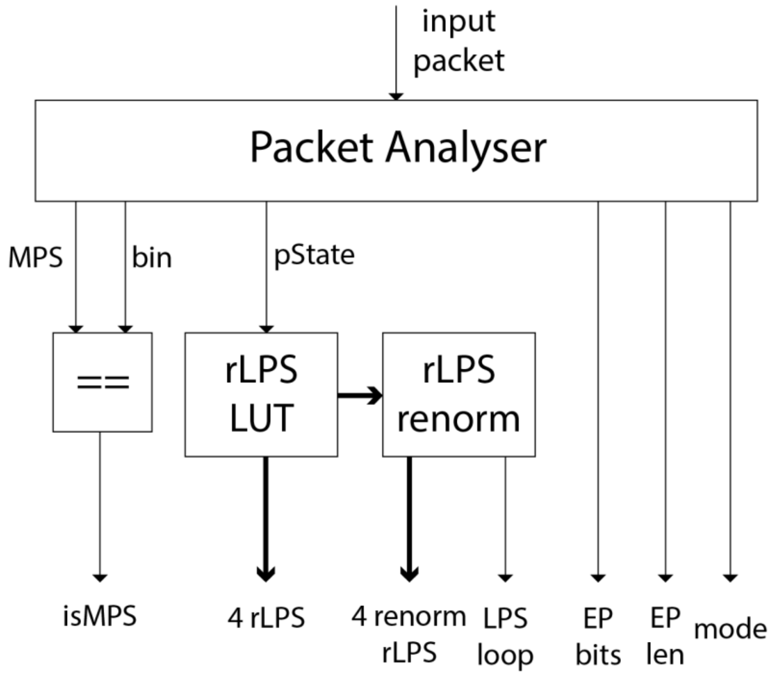


Figure 17: BAE stage 1 architecture.

The input of this stage is a 10-bit packet with a specific format for each bin:

- Regular/terminate bin: mode identification *mode* (2 bits), bin value *bin* (1 bit), probability state *pState* (6 bits) and MPS value *MPS* (1 bit)

- Bypass bin: mode identification *mode* (2 bits), bypass bin value *EPbits* (4 bits), the number of bypass bins *EPlen* (3 bits) and one padding 0.

This format allows bypass packet to contain up to 4 bypass bins.

Bin value is compared with MPS value and four *rLPS* candidates are produced from *rLPS* look-up table based on pState.

In stage 2, *rLPS* and *rMPS* are renormalized before one of them is chosen to assign to the renormalized Range. However, *rMPS* renormalization is much simpler than *rLPS*'s. Because Range is always bigger than 256 and rMPS is larger than *rLPS, rMPS* certainly exceeds 158. Thus, *rMPS* renormalization happens for one round or even not at all. Whereas, *rLPS* renormalization occurs for at most 7 rounds. That significantly contributes to the critical path of stage 2. While stage 1's task is rather trivial, moving *rLPS* renormalization to this stage seems logical.

All of the four *rLPS* candidates are renormalized in the first stage. *rLPS* renormalization includes two steps: specifying the number of rounds and shifting *rLPS* one bit to the left for that number of times. We determine the amount of renormalization cycles *LPSloop* by using a look-up table provided by Marpe *et al*. [20]. Then a cascade of selective shifters shifts the range by the specified amount.

4.2.2. Range Update

Stage 2 is dedicated to interval subdivision and the update of *Range*.

The bottom register feeds back *Range* value to the stage, which uses to determine *rLPS* and renormalized *rLPS* from the four candidates. Then interval subdivision is performed by subtracting *rLPS* from *Range*.

rMPS is renormalized for 0 or 1 round based on its MSB. If the MSB is '1', it is indicated that *rMPS* is bigger or equal to 256, and rescaling is not necessary. Otherwise, *rMPS* is doubled. The updated Range is picked from the renormalized *rMPS* or *rLPS*.

In bypass mode, the task for stage 2 is minimal. *Range* is be kept unchanged and the number of renormalization cycle is equal to *EPlen*. Thus, we place some multiplexers at the end to enable this mode. Besides, in stage 3, it requires the multiplication of *Range* and *EPbits* to calculate renormalized *Low*. Due to the unchanged *Range* in bypass mode, the multiplication can occur at the beginning of this stage. Therefore, placing the multiplier in stage 2 instead of stage 3 would obviously lower the latency of stage 3 while cause no effect to stage 2.

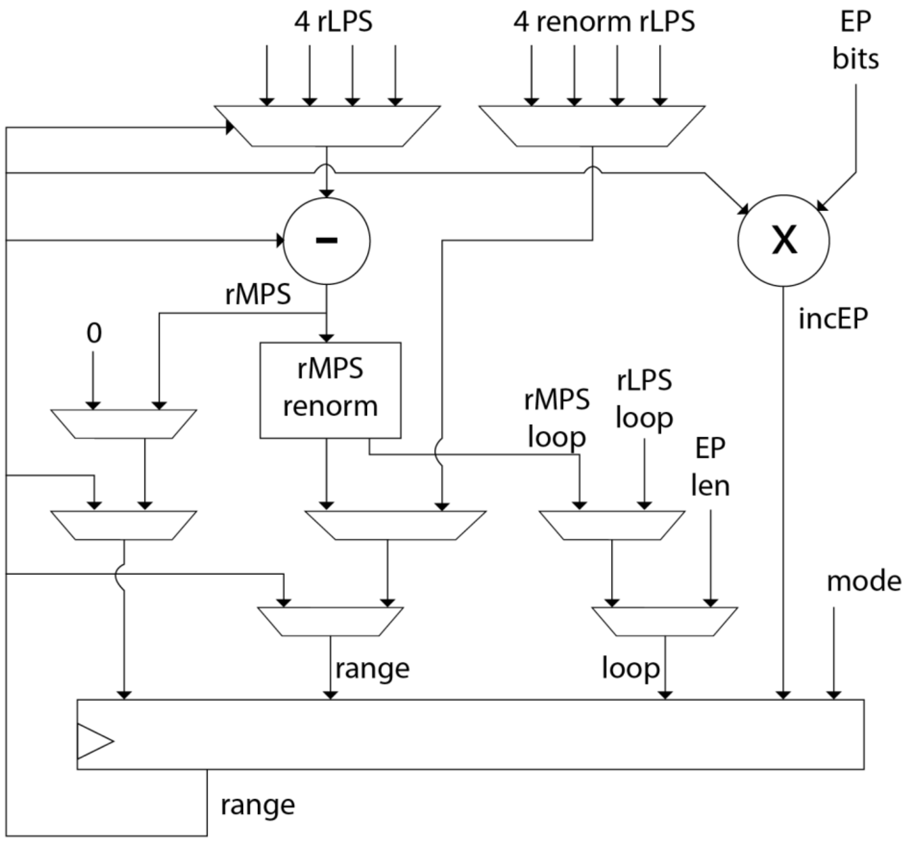


Figure 18: BAE stage 2 architecture.

4.2.3. Low Update

Low renormalization is conducted at this stage.

Calculating renormalized *Low* is different for each type of bin. For regular or terminate bin, *Low* increases an amount of *rMPS* if the bin is LPS, then it is shifted left by the number of bits that is defined in *Range* renormalization. The formula implemented in our hardware is:

$$\mathrm{renormLow} = \mathrm{rMPS} \ll \mathrm{loop} + \mathrm{oldLow} \ll \mathrm{loop}$$

loop and *rMPS* are determined in the second stage. *loop* is *rMPSloop* if the bin is MPS, or *rLPSloop* otherwise. *rMPS* is set to 0 in the case of MPS.

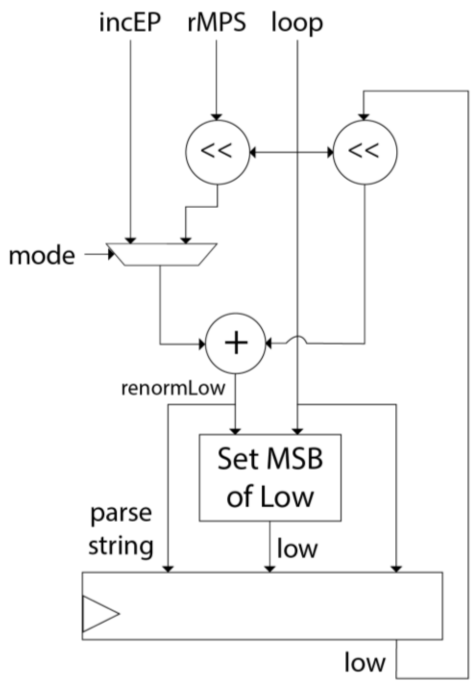


Figure 19: BAE stage 3 architecture.

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

$$\mathrm{renormLow} = \mathrm{Range} \times \mathrm{binVal} + \mathrm{oldLow} \ll 1$$

As the first term is *Range* when bin is ‘1’ and 0 when bin is ‘0’, it is convenient to employ a multiplier. For the next bypass bin, *Range* remains the same and *Low* is doubled. Therefore, after encoding a string of bypass bin, the term is the multiplication of *Range* and the string. In our design, the formula for this process is:

$$\mathrm{renormLow} = \mathrm{rMPS} \times \mathrm{EPbits} + \mathrm{oldLow} \ll \mathrm{loop}$$

rMPS and *loop* are set to *Range* and *EPlen*, respectively, in the previous stage. *EPlen* records the number of bypass bins being processed, which can range from 1 to 4. Since *EPbits* contains a string of bypass bins, it is the multiplier that enables to process all of those bins at once.

Unlike *Range* when the amount of shift is just enough to push it above the lower bound 256 but is still representable with 9 bits, renormalized *Low* is beyond the 10-bit representation limit. As depicted in figure 6, in each renormalization cycle, *Low* might be reduced before being doubled. Thus it demands a particular mechanism to extract the updated *Low* from the renormalized *Low*. *newLow*[8 : 0] is assigned to *renormLow*[8 : 0]. In order to determine the MSB of *newLow*, we examine the upper bits of the renormalized *Low* whose index is bigger than 8, *renormLow*[9 + loop : 9], which is referred as the parsing area. If the parsing area is a sequence of '1' bits then *newLow*[9] = 1, otherwise, it is 0.

4.2.4. Bitstream Generator

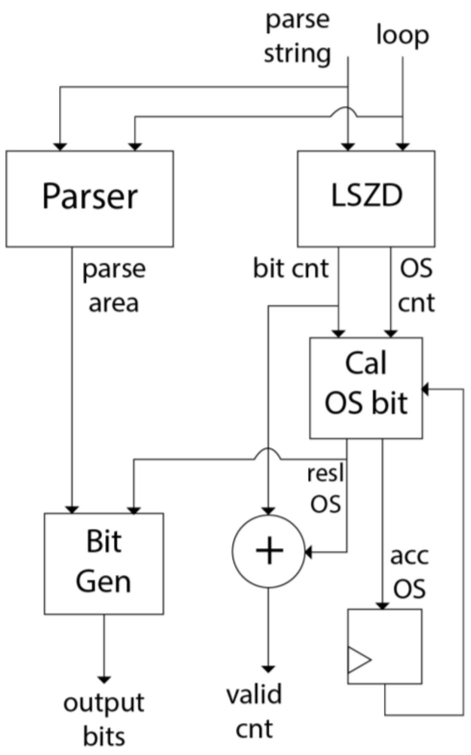


Figure 20: BAE stage 4 architecture.

Least Significant Zero Detector checks the parsing area for the position of the last zero. The detected position corresponds to the number of outstanding bits accumulated after this cycle. Those bits on the left of the spotted zero bit are confirmed output bits that will be produced at the end of this stage. We implemented a table-based LSZD to accomplish this task.

Due to the variable length of the parsing area, it is inevitable to align it with the parse string. Thus, Parser, consists of a set of shifters, is responsible for shifting *ParseString* so that *ParseArea* appears at the most upper bits.

The number of resolved outstanding bits is calculated based on the number of confirmed output bits *bitCnt* and the number of newly detected outstanding bits *OSCnt*. Outstanding bits found in this cycle are always saved for later. If there are no determined output bits, no outstanding bits is resolved and all of them are accumulated. Otherwise, outstanding bits from the previous cycle are confirmed and the number of accumulated outstanding bits equals *OSCnt*.

validCnt records the number of valid bits in the output codeword. It is the total of confirmed output bits and resolved outstanding bits.

Bit generator produces a 38-bit codeword at each cycle. The format of the output includes four parts:

- MSB of the parsing area (1 bit)
- Resolved outstanding bits (0 – 31 bits)
- Remaining bits of Parse Area (6 bit)
- Padding ‘0’ bits (0 – 31 bits)

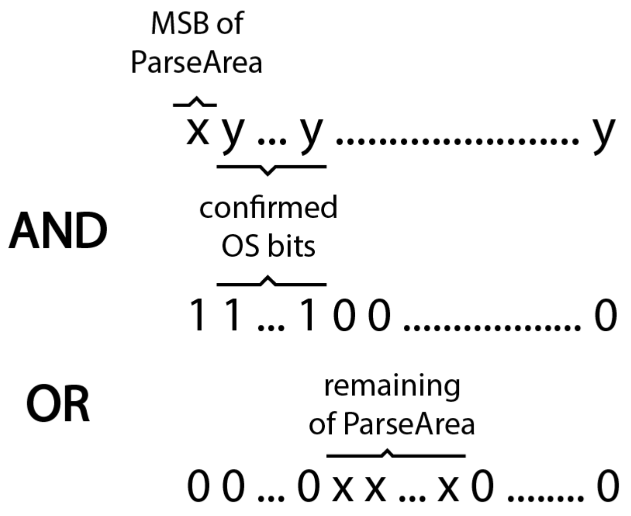


Figure 21: Bit generation mechanism.

To form the codeword, we used layering technique. The four parts are gradually added to the codeword through each layer. Due to the unfixed number of resolved outstanding bits, shifter is utilized to create the conformable layers.

A significant portion of our one-core architecture shared by all types of bin and the small number of pipeline stages can result in marginal area usage. We reorganized each stage to keep the balance between them and shorten the critical path in burdensome stages, which predicts an acceptable operating frequency. Therefore, our design can guarantee low power consumption. The next chapter will present the implementation result that will prove the soundness of our proposals.

IMPLEMENTATION RESULT

After designing the architecture, we set up a testing environment to verify the design and then implemented with a synthesis tool. The detail of the verification method and the implementation result will be discussed in the following sections.

5.1. Verification Method

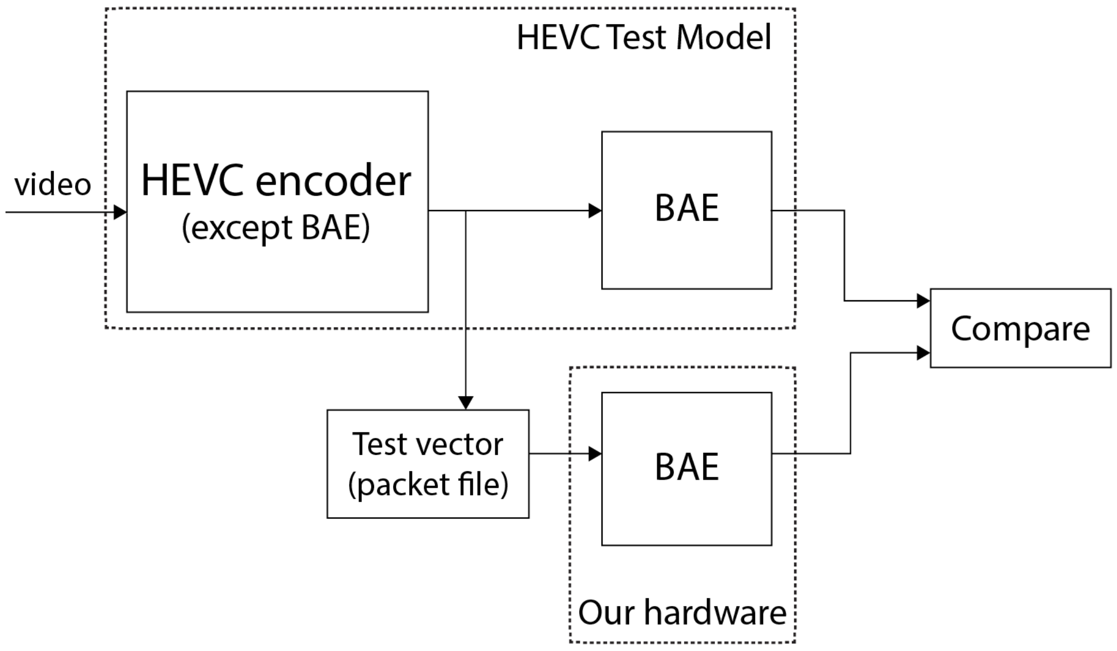
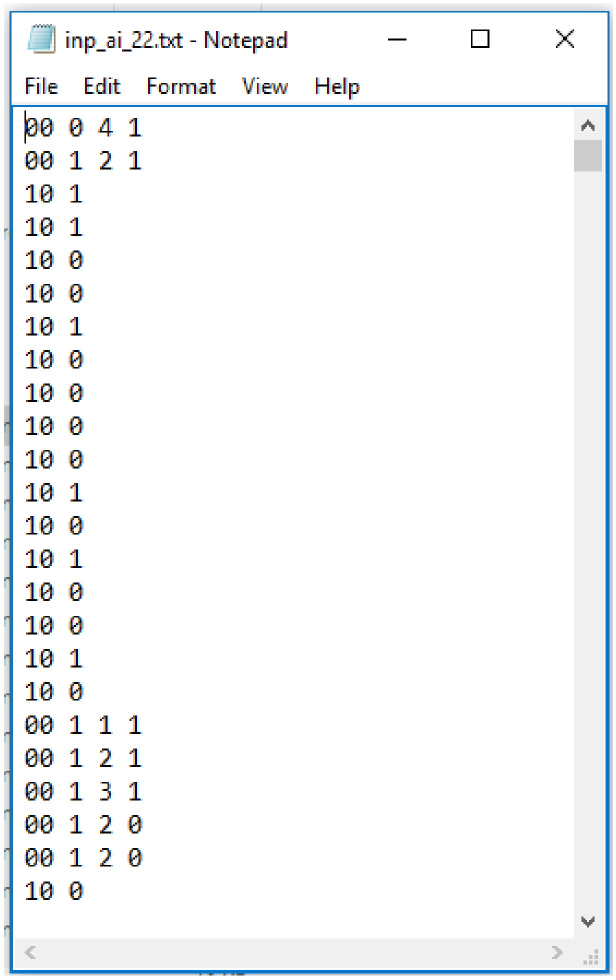


Figure 22: Our verification method.

HEVC Test Model (HM) [17] is the official reference software, written in C++, of the HEVC video coding standard. It is a working prototype that includes the tools or algorithms, which aims at testing and development of the standard. This work heavily relied on the software for understanding the algorithm, developing the initial ideas and conducting the testing strategy described in figure 22.

In this work, we used HM version 16.12. A CIF video test sequence was encoded by HM with different configurations and quantization parameters. This test sequence covers all the input circumstances of BAE.



```
00 0 4 1
00 1 2 1
10 1
10 1
10 0
10 0
10 1
10 0
10 0
10 0
10 0
10 1
10 0
10 1
10 0
10 0
10 1
10 0
00 1 1 1
00 1 2 1
00 1 3 1
00 1 2 0
00 1 2 0
10 0
```

Figure 23: Input file extracted from HM.

With some adjustment to the software, we were able to extract the input and output of Binary Arithmetic Encode in two separated text files. One file contains output bytes of BAE. And the other file includes input vectors for BAE with formats:

- Regular bin: mode (00), bin value, MPS value, probability state
- Bypass bin: mode (10), bin value
- Terminate bin: mode (01), bin value

A Python program was written to pack these integer values into a binary string that match the input packet format of our BAE architecture. We also wrote a Python program to model our proposed architecture's function before designing with hardware description language (HDL).

Our hardware design was implemented in VHDL. The model read the input packet file and generated a text file that consists of output bytes. Then it was compared with the output extracted from HM by using a Python program. Any differences would indicate errors in our design. Thus, the hardware model was edited until the two files are the same.

5.2. Implementation Result

We tested our BAE architecture under several test cases. All-intra or low-delay configurations with the quantization parameter of 22 and 37 was configured in the reference software, which created four different sets of test vectors.

The design was simulated in Questa 10.2c. The result is shown in figure 24 and 25.

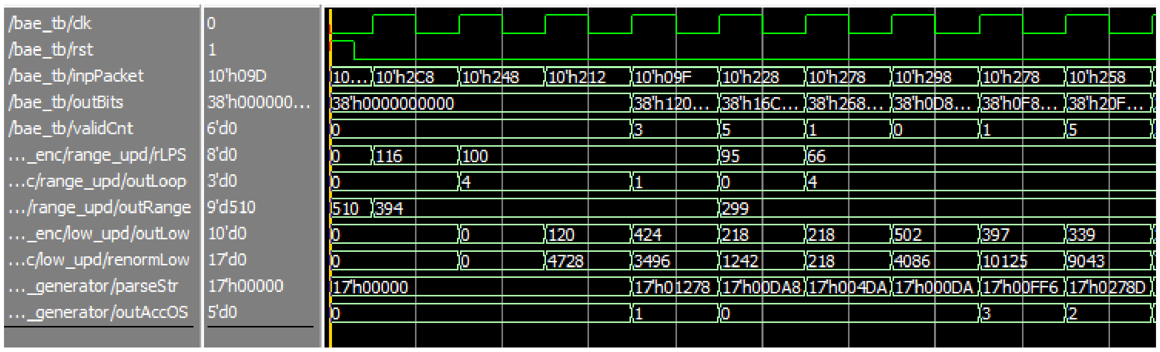
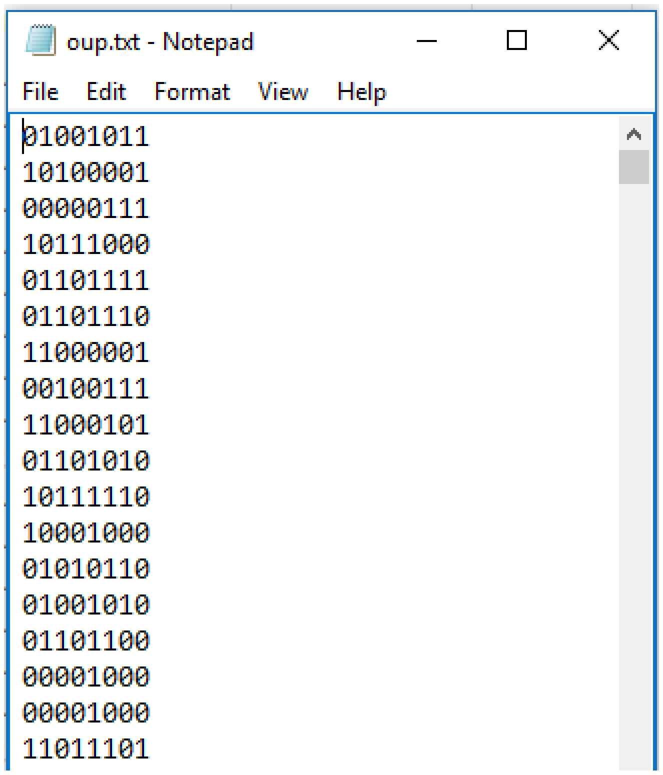

/bae_tb/clk 0
/bae_tb/rst 1
/bae_tb/inpPacket 10'h09D
/bae_tb/outBits 38'h000000...
/bae_tb/validCnt 6'd0
..._enc/range_upd/rLPS 8'd0
...c/range_upd/outLoop 3'd0
.../range_upd/outRange 9'd510
..._enc/low_upd/outLow 10'd0
...c/low_upd/renormLow 17'd0
..._generator/parseStr 17'h00000
..._generator/outAccOS 5'd0
10... 10'h2C8 10'h248 10'h212 10'h09F 10'h228 10'h278 10'h298 10'h278 10'h258
38'h0000000000 38'h120... 38'h16C... 38'h268... 38'h0D8... 38'h0F8... 38'h20F...
0 3 5 1 0 1 5
0 116 100 95 66
0 4 1 0 4
510 394 299
0 0 120 424 218 218 502 397 339
0 0 4728 3496 1242 218 4086 10125 9043
17'h00000 17'h01278 17'h00DA8 17'h004DA 17'h000DA 17'h00FF6 17'h0278D
0 1 0 3 2

Figure 24: Simulation waveform.

To guarantee the behavior of a 4-stage pipelined architecture, we set the initial value for each parameter so that nothing will be outputted in the first three cycles. Valid output appears from the fourth cycle.



```
01001011
10100001
00000111
10111000
01101111
01101110
11000001
00100111
11000101
01101010
10111110
10001000
01010110
01001010
01101100
00001000
00001000
11011101
```

Figure 25: Simulation output file.

Table 3: 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

With the ability to process at most 4 bypass bins in a clock cycle, our BAE achieved the performance of 1.4 bins per cycle in average.

The RTL model was synthesized with Synopsys Design Compiler in NANGATE 45 nm process technology.

Table 4: Delay in each pipeline stage

Stage	**Latency (ns)**
rLPS LUT	1.08
Range update	1.25
Low update	1.69
Bit gen	1.52

The worst case latency is 1.69 ns in the third stage. This is due to the cascade of a series of shifters and a 16-bit adder presented at this stage. Although the computing task of stage 2 is considered as the biggest, placing the multiplier in parallel with other submodules here doesn’t affect the latency.

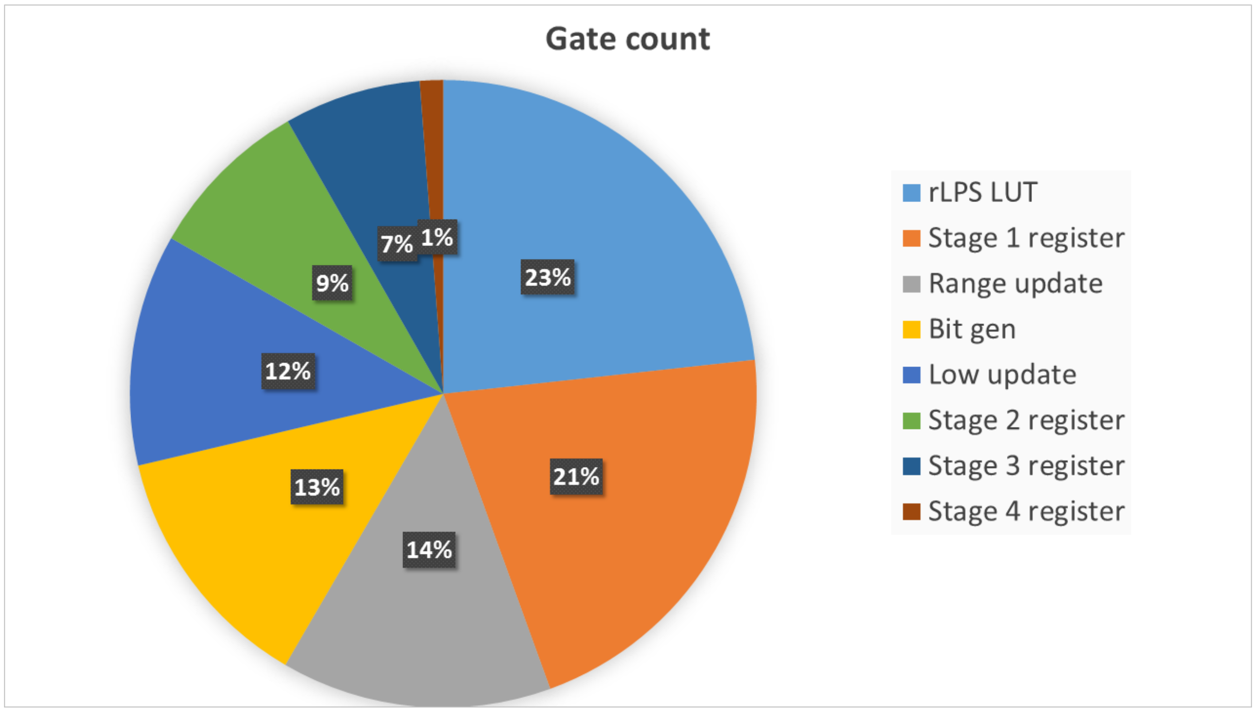


Figure 26: Pie chart of area cost.

The total gate count of our hardware core is 2.83 K gates. Stage 1 has the highest gate count owing to a large amount of utilized memory. It contains two LUTs, one for rLPS and the other for the number of rLPS renormalization cycles. The register after stage 1 also takes up vast space due to the 93-bit input/output port, equivalent to 93 register cells.

Table 5: Power dissipation analysis

Power group	**Internal power (uW)**	**Switching power (uW)**	**Leakage power (uW)**	**Total power (uW)**
Sequential	107.18	5.55	13.94	126.68 (35.62 %)
Combinational	75.95	120.65	32.32	228.92 (64.38 %)
Total	183.13	126.2	46.26	355.6

The recorded power consumption is 0.36 mW. In comparison with other works, our work achieves an acceptable throughput with the smallest area cost and power dissipation.

Table 6: Comparison with other works

	Zhou2014 [15]	Pham2014 [13]	Jo2016 [14]	Ramos2016 [18]	**Our work**
Bins per cycle	4.37	1	1.99	4	**1.4**
Clock frequency (MHz)	420	180	1110	280	**591.7**
Technology process (nm)	90	180	65	45	**45**
Throughput (Mbin/s)	1836	180	2219	1120	**828**
Gate count (K gates)	NA	3.96	5.68	9.95	**2.8**
Power consumption (mW)	NA	2.88	NA	2.49	**0.36**

[15], [14] and [18] use many pipelined BAE cores to process multiple bins. [13] uses 3 custom cores for 3 types of bins, which is quite expensive and not really efficient, while we use only one pipelined core for all types. That explains our minimal gate count compared to others.

The NANGATE 45 nm technology provides a superior power saving feature. Although [18] uses the same technology, the usage of 4 parallel pipelined cores, which gives four times larger gate count and bigger throughput than our work’s, leads to higher power consumption.

The obtained result proved our effort toward a low-cost, low-power BAE for HEVC CABAC. With the throughput of 828 Mbin/s, our design is capable of encoding in real-time a video conforming to the main profile level 6.1 of high tier [8], which equivalent to a 4K video with the frame rate of 256 fps.

CONCLUSIONS

6.1. Conclusions

Binary Arithmetic Encoder is the most crucial component of CABAC that expresses the main function of the module, entropy coding. In the rising of the Internet of Things industry, cost- and energy-effective hardware is highly demanded. This thesis proposed a low-cost, low-power Binary Arithmetic Encoder for HEVC CABAC. An overall architecture of CABAC encoder was also presented. The simplicity of bypass encoding was exploited to enable multiple bins processing. That led to the feature of encoding up to 4 bypass bins per clock cycle. Furthermore, the 4-stage pipelined architecture, which can process all types of input in one core and have each stage rearranged, significantly reduced the critical path and hardware cost. This design met the requirement of real-time processing at main profile level 6.1 of high tier.

6.2. Future Works

In the future, we will optimize several submodules in the design and consider some advanced techniques such as multiple cores for BAE to maximize the throughput. We will also design architecture of Context Modeler and Binarizer to form a completed design of CABAC encoder.

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] I. E. Richardson, The H.264 Advanced Video Compression Standard, John Wiley & Sons, 2010.

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

[6] 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.

[7] M. Wien, High Efficiency Video Coding: Coding Tools and Specification, Springer-Verlag Berlin Heidelberg, 2015.

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

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] "HEVC Test Model, HM 16.12," [Online]. Available: https://hevc.hhi.fraunhofer.de/trac/hevc/browser/tags/HM-16.12.

[18] 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.