Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

High throughput VLSI architecture for H.264/AVC context-based adaptive binary arithmetic coding (CABAC) decoding*

Kai HUANG†1, De MA1, Rong-jie YAN†‡2, Hai-tong GE3, Xiao-lang YAN1
(1Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China)
(2State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
(3Hangzhou C-Sky Micro-System Company, Hangzhou 310012, China)
†E-mail: huangk@vlsi.zju.edu.cn; yrj@ios.ac.cn

Received Aug. 28, 2012; Revision accepted Mar. 6, 2013; Crosschecked May 13, 2013

Abstract: Context-based adaptive binary arithmetic coding (CABAC) is the major entropy-coding algorithm employed in H.264/AVC. In this paper, we present a new VLSI architecture design for an H.264/AVC CABAC decoder, which optimizes both decode decision and decode bypass engines for high throughput, and improves context model allocation for efficient external memory access. Based on the fact that the most possible symbol (MPS) branch is much simpler than the least possible symbol (LPS) branch, a newly organized decode decision engine consisting of two serially concatenated MPS branches and one LPS branch is proposed to achieve better parallelism at lower timing path cost. A look-ahead context index (ctxIdx) calculation mechanism is designed to provide the context model for the second MPS branch. A head-zero detector is proposed to improve the performance of the decode bypass engine according to UEG*k* encoding features. In addition, to lower the frequency of memory access, we reorganize the context models in external memory and use three circular buffers to cache the context models, neighboring information, and bit stream, respectively. A pre-fetching mechanism with a prediction scheme is adopted to load the corresponding content to a circular buffer to hide external memory latency. Experimental results show that our design can operate at 250 MHz with a 20.71k gate count in SMIC18 silicon technology, and that it achieves an average data decoding rate of 1.5 bins/cycle.

Key words: H.264/AVC, Context-based adaptive binary arithmetic coding (CABAC), Decoder, VLSI
doi:10.1631/jzus.C1200250 **Document code:** A **CLC number:** TN919.8

1 Introduction

The ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG) jointly developed the latest video standard H.264/AVC (ITU-T Recommendation H.264:2003) for next generation multimedia coding applications. Compared to existing video coding standards like H.263 or MPEG-4, H.264/AVC provides more than twice the compression ratio while maintaining video coding quality. The gain in coding efficiency is due mainly to the adoption of many new techniques, such as multiple reference frames, weighted prediction, deblocking filtering, and context-based adaptive entropy coding. There are two approaches for context-based adaptive entropy coding: context-based adaptive variable length coding (CAVLC) for both baseline and main profiles, and context-based adaptive binary arithmetic coding (CABAC) for main profiles only. Compared with CAVLC, CABAC achieves better compression efficiency, but it brings higher computation complexity during decoding (Shi *et al.*, 2008). Statistical results show that it takes 30–40 cycles to decode a single bin on a DSP processor (Yu and He, 2005). In the case of 1080P main profile

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 61100074) and the Fundamental Research Funds for the Central Universities, China (No. 2013QNA5008)

video decoding, the throughput of a video coder using CABAC reaches almost 150 Mbin/s, which makes it difficult to implement in a programmable processor. Therefore, an efficient hardware decoder is important for low-power and real-time H.264 codec applications.

The decoding process of CABAC is bit-serial and has strong data dependency because the decoding result of one bit always has a direct effect on the next bin process. This dependency makes it difficult to exploit parallelism during decoding. What is worse, the context models of the current syntax element (SE) are closely related to the results of its neighboring macroblocks (MBs) or blocks, which leads to frequent memory access. Research to address these issues has focused mainly on two aspects: parallelism exploitation and memory access optimization. To improve parallelism, most solutions concatenate multiple decode engines to decode multiple bins in one cycle. In previous designs (Yang *et al.*, 2006; Yang and Guo, 2009), two- or four-bin decoding parallelism is exploited in one cycle by using a look-ahead decision parsing scheme for the decode decision engine and a dual-series bypass parsing scheme for the decode bypass engine. However, the parallelisms are implemented simply by concatenating two decode decision engines and four bypass decode engines, which extends the timing critical path of the hardware circuit and restricts the CABAC decoder clock frequency. Another solution (Yuan, 2008) is to employ a three-stage pipeline structure to explore the decoding parallelism. The pipeline has to be stalled when data dependency occurs between bin i+1 and bin i. To avoid data hazards, some prediction-based pipelined architectures (Shi *et al.*, 2008; Kuo *et al.*, 2011; Liao *et al.*, 2012) have been proposed to achieve high-throughput. Some methods, such as syntax element prediction, redundant circuits, and forwarding techniques, can be adopted to avoid pipeline stalls. However, the design of Shi *et al.* (2008) does not utilize the memory bandwidth well and each pipeline stage contains at least one memory access, which greatly increases the frequency of memory access. Moreover, the decoder has to load two context models and store one in every cycle. Thus, memory access conflicts occur frequently and two dual-port static random access memory (SRAM) devices have to be used to solve them, which increases the cost of hardware. Although the design of Liao *et al.* (2012) can decode in high-rate mode, almost all context models are stored on chips and both dual-port SRAM and registers are used, which impose heavy hardware costs on the gate count. Kuo *et al.* (2011) proposed an area-efficient architecture, but only a single-bin engine is used and the throughput is low.

Since the designs of Yang *et al.* (2006), Chen and Lin (2007), Yuan (2008), and Yang and Guo (2009) can decode more than two bins per cycle, an efficient memory architecture for context models is proposed to eliminate the redundant latency for context accessing. Yang *et al.* (2006) adopted a pair of ping-pang cache registers to improve data reusability. Chen and Lin (2007) divided the context table into two memory tables to be read concurrently to improve data throughput. Both architectures need to manage two memory blocks. Yuan (2008) rearranged the context models to place the models used by the same SE at the same memory address to form a cluster, and load one cluster into a register buffer each time. However, to load one cluster in each cycle, the data width has to be at least 105 bits, which is difficult to implement in a small memory. Also, two clusters have to be frequently loaded in turn if some SEs are decoded alternately, i.e., significant_coeff_flag and last_significant_coeff_flag.

This paper presents a new architecture design of an H.264/AVC CABAC decoder which rearranges the context table memory to improve memory efficiency and reduce hardware cost, and which optimizes both the decode decision and decode bypass engines to increase parallelism with a reduced timing penalty. We have reorganized the context table into 29 groups to ensure that each group is loaded only once during the decoding process of one MB, and have adopted a 112-bit circular buffer to cache the context models. Both of these changes reduce memory bandwidth dramatically. Furthermore, we have divided the decode decision engine into two half branches: the most possible symbol (MPS) decode decision branch and the least possible symbol (LPS) decode decision branch. The MPS branch is much simpler than the LPS branch. Therefore, two MPS branches are sophisticatedly concatenated to decode two bins in one cycle and the critical path is kept almost the same as that of the decode decision engine. To provide a context model for the second MPS branch, we propose a look-ahead context index (ctxIdx) calculation mechanism. The decode bypass engine is used mainly to decode two SEs, motion vector difference (MVD) and coeff_abs_level_minus1, encoded as UEG0 and UEG3, respectively. According to the features of UEG*k*, the suffix of the two SEs starts with a series of 1's decoded by the LPS branch of the decode bypass engine, which means that only the LPS branch will be used at the first decoding. Therefore, we have merged four LPS branches to decode four bins in one cycle and then used two series of decode bypass engines to decode the remaining bins. Also, we have classified SEs into four categories and employed different strategies to decode each category. The proposed design can achieve from 1 to 2 bins per cycle for the decode decision engine, and from 2 to 4 bins per cycle for the decode bypass engine.

2 Overview of CABAC decoding flow

Fig. 1 shows the CABAC decoding flow. Before decoding the first SE in the slice data, both the context table and CABAC decoding engine should be initialized according to the value of slice QPy and the first nine bits from the bit stream. The SEs of each MB can then be decoded sequentially. The decoding flow of each SE consists of three fundamental steps: context model selection, bin decision, and inverse binarization.

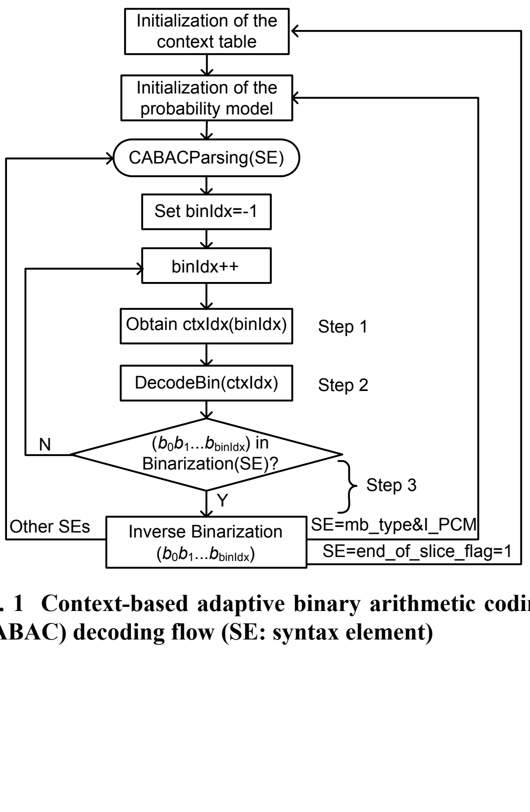
Initialization of the context table
Initialization of the probability model
CABACParsing(SE)
Set binIdx=-1
binIdx++
Obtain ctxIdx(binIdx)
Step 1
DecodeBin(ctxIdx)
Step 2
N
$(b_0b_1...b_{\text{binIdx}})$ in Binarization(SE)?
Step 3
Y
Other SEs
Inverse Binarization $(b_0b_1...b_{\text{binIdx}})$
SE=mb_type&I_PCM
SE=end_of_slice_flag=1

Fig. 1 Context-based adaptive binary arithmetic coding (CABAC) decoding flow (SE: syntax element)

A context model is a probability model associated with one or more bins of each SE. It is used to store the MPS value and the parameter pStateIdx, which is used to index the range of the LPS values. Before deciding each bin value of an SE, the context model index (ctxIdx) should be calculated using the information of neighboring blocks or previously decoded bin values, and then the corresponding MPS value and pStateIdx values will be loaded from the memory of context models. Continuous updating of context models during the whole decoding process causes frequent memory access. Therefore, efficient managing of context models can reduce memory bandwidth and also improve the CABAC decoding speed.

The bin decision flow consists of three decoding engines, the decode decision engine, decode bypass engine, and decode termination engine. Only one of these engines is used while deciding one bin value according to the SE and the type of bin (Fig. 2). The complexity of each engine is different and each engine has a different effect on the context model, codIRange and codIOffset respectively. The termination engine is used only to decode end_of_slice_flag and the bin in I_PCM mode. It can also be implemented using the decode decision engine with ctxIdx=276. In this study, we focus mainly on the decode decision engine and decode bypass engine.

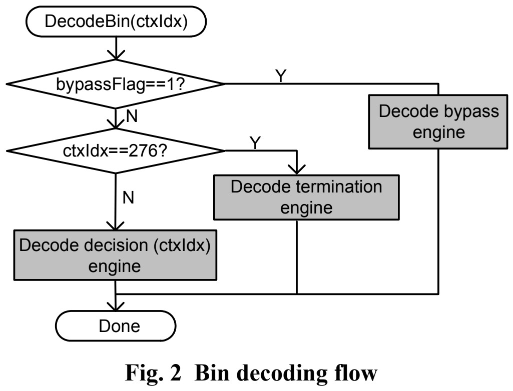

Fig. 2 Bin decoding flow

Each SE value is binarized according to certain rules associated with the type of SE. When the accumulated decoded bin string conforms to one binarized value of an SE, the decoding process of the corresponding SE is over and the bin string is inversely binarized to generate the value of the SE according to the binarized rules.

3 Optimization techniques

To decode one bin, the corresponding context model should first be loaded and then the updated value has to be written back. Both actions increase the frequency of memory access. The decoding process of each bin needs to call one of the three arithmetic decoding engines. The decode decision engine, used for most SEs, is quite complex and thus becomes the critical timing path of VLSI design. To improve the throughput of bin decoding, several engines are sometimes concatenated (Yang and Guo, 2009), which makes the critical timing path even longer. Based on this analysis, the bottlenecks for efficient CABAC decoding are the maintenance of context models and arithmetic decoding. In this section, we analyze the characteristics of CABAC decoding and present the techniques used in our VLSI architecture. Some terms used in the rest of the paper are defined as follows: codIRange is the width of the current interval, codIOffset is the position of the current bin at codIRange, rMPS is the estimated probability interval of the MPS, and rLPS is the estimated probability interval of the LPS.

3.1 Arithmetic engine division and reorganization

Fig. 3 shows the workflow of the decode decision engine. It consists of four main parts, range calculation (RanCal), LPS decoding branch (LPSDecB), MPS decoding branch (MPSDecB), and renormalization (Renorm). We find that the complexity of MPSDecB is much lower than that of LPSDecB. When the LPS branch is selected, both codOffset and valMPS are re-calculated before the renormalization stage, while the MPS branch does nothing with these two values. According to the H.264/AVC standard (ITU-T Recommendation H.264:2003), the highest value of rLPS is 240 and the lowest value of codIRange before decoding is 256, which means that the value of rMPS is not less than 16 (rMPS=codIRange −rLPS). Therefore, when the MPS branch is selected, the renormalization process is quite simple and needs to shift only 4 bits at most, while the LPS branch needs to shift 7 bits. According to our analysis, the renormalization process of LPSDecB is about twice as slow as that of MPSDecB. We can decode two bins if both continuous bins select the MPS branch and keep the decoding critical timing path the same as that of the decode decision engine.

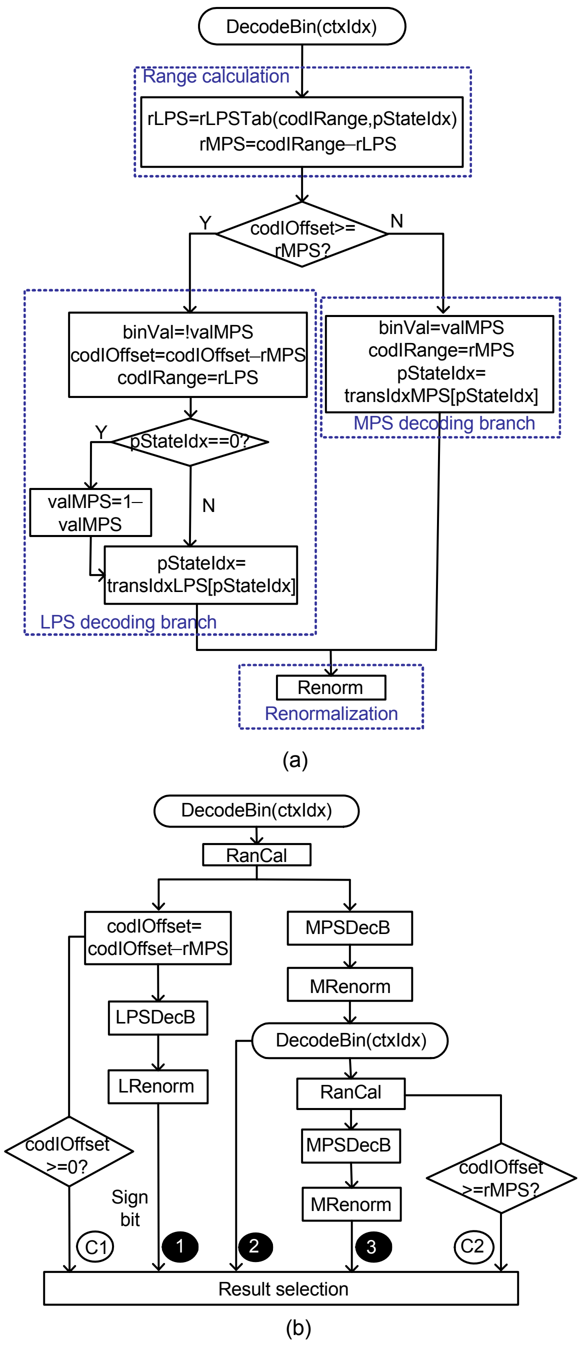

Fig. 3 Workflow of the decode decision engine (a) and the new engine (b)

To employ the simple operation of the MPS branch and to decode two bins in certain conditions, we divide the decode decision engine into four parts, RanCal, LPSDecB, MPSDecB, and Renorm (Fig. 3a) and reorganize these four parts into a new decode decision engine. Fig. 3b illustrates the workflow of the new engine. The LPS branches and the two series of MPS branches are designed to be processed in parallel. Two comparison results between codIOffset and rMPS are used to select the decoding results. When comparison results 'C1' and 'C2' are both 'No', the decoding results '2' and '3' are valid; otherwise, only one bin is decoded from either result '1' or '2'. From Fig. 3b, we can see that two ctxIdxs should be calculated to decode two bins in each cycle for the right branch. To solve this problem, we propose a look-ahead ctxIdx calculation mechanism. The working mechanism of the look-ahead ctxIdx calculation logic is explained in Section 3.3.

3.2 Improvement of context module buffer access

Many memory access operations are involved in decision decoding. During the process of decoding each bin, the corresponding context model should be loaded and its updated value has to be written back. If we want to decode more than one bin per cycle, the memory architecture for accessing context modes should be designed more efficiently. It takes at least two cycles to load the context model from random access memory (RAM) and write the updated value back to RAM to maintain one context model. Therefore, we have reorganized the entire table of context models for H.264, decoding into 29 groups according to the decoding characteristics of the SEs, and each group contains at most 16 context models. Each context model comprises a 6-bit pStateIdx and a 1-bit MPS. Moreover, a 112-bit circular register buffer is designed to read and write context memory in a more flexible way. At the start of decoding, the first group of context models is loaded into the circular buffer. When more than 28 bits (corresponding to a word address in the memory of the context model) need to be recycled, the new value of the corresponding context models will be updated into memory, and the next group will be loaded to fill the empty space. The circular buffer is maintained like an infinitely large memory for the context model, which greatly improves the memory access efficiency of the decode engine.

Compared with the previous approach of context model rearrangement (Yuan 2008), which causes some groups to be loaded frequently, we have adopted three techniques to ensure that each group is updated only once during the decoding process of these SEs in one MB:

1. The context models of one SE are divided into different memory groups according to different slice types. A total of 24 context models are used for syntax element 'mb_type' (eight, seven, and nine models are used for the I slice, P slice, and B slice, respectively). Using this technique, only the necessary memory group is loaded for a given slice type. We can thereby reduce memory access operations for context models as much as possible and make full use of the loaded data.

2. The context models of continuous SEs are combined into one group until there are 16 context models. For example, some SEs like mb_type of I slice, intra_chroma_pred_mode, prev_intra4×4_pred_mode_flag, and rem_intra4×4_pred_mode_flag, appear continuously in the bit streams and can be allocated to the same group. This technique ensures that each group needs to be loaded and written back only once during decoding one MB.

3. SEs that appear in pairs are combined into one group. The context models with ctxIdx from 105 to 165 are all used for the SE significant_coeff_flag while those with ctxIdx from 166 to 212 are used for SE last_significant_coeff_flag. The two kinds of SEs appear alternately in the bit streams and use the same method to calculate their ctxIdx. We organize the corresponding context models of these SEs into one group; e.g., the context models of SE significant_coeff_flag with ctxIdx 105–112 and those of SE last_significant_coeff_flag with ctxIdx 166–173 form one group.

With these techniques, the frequency of access to the memory of the context models is greatly reduced. Compared to the designs of Shi *et al*. (2008) and Yuan (2008), the frequency of memory access is reduced by almost 70% and 30%, respectively. Taking advantage of the circular buffer, more than one model can be referred to at the same time, making it possible to decode two bins in one cycle.

3.3 Division of the syntax element

Before decoding each bin, it is necessary to calculate ctxIdx (Fig. 1). To decode two bins per cycle when both are MPS, it should be able to calculate two ctxIdxs in certain conditions. However, the method to calculate ctxIdx depends largely on the type of SE, and even the previous decoded bin. In some cases, even if two continuous MPSs occur, the

SE does not need to decode two bins at a time; i.e., mb_skip_flag is a 1-bin SE. According to the analysis, we divide the SEs into four categories and an appropriate ctxIdx calculation strategy is adopted according to the category type. Table 1 shows the four categories of SE.

Table 1 Category of the syntax elements

Group ID	Group description	Related syntax element
1	SE has only one bin	mb_skip_flag, mb_field_decoding_flag, coeff_sign_flag, end_of_slice_flag, coded_block_flag
2	The ctxIdx of the current bin depends on the value of the previous bin	mb_type(I, P, B), sub_mb_type(B), coded_block_pattern
3	Two one-bin SEs appear in pair	prev_intra4×4_pred_mode_flag, rem_intra4×4_prev_mode_flag, significant_coeff_flag, last_significant_coeff_flag
4	Others	sub_mb_type(I, P), mvd_*x*, mvd_*y*, ref_idx_l0, ref_idx_l1, mb_qp_delta, intra_chroma_pred_mode

1. One-bin SE

In this condition, we need to calculate only one ctxIdx. The other ctxIdx calculation logic and the second MPS decoding logic (MPSDecB) (Fig. 3b) should be disabled for power saving. The decoded value is chosen only from values '1' and '2' according to the comparison result 'C1'.

2. SE depending on the previous bin

To decode two bins in each cycle, we propose a look-ahead ctxIdx calculation scheme for these SEs. For the SEs in this category, the ctxIdx of the second bin depends on the value of the first bin and the second MPSDecB is valid only when the MPS occurs in the first one. So we calculate the ctxIdx of the second bin in advance on the assumption that the first bin hits the MPS condition.

3. SEs appearing in pairs

For this SE category, the second SE needs to be decoded only when the previous SE value is equal to the designated value. Therefore, the second bin's SE type depends largely on the value of the previous bin. For example, the SE last_significant_coeff_flag needs to be decoded only when the previous SE significant_coeff_flag is equal to 1. When significant_coeff_flag is 0, the next SE is also significant_coeff_flag. In this condition, the look-ahead ctxIdx calculation scheme works in the same way as we decide the SE type of the next bin according to the current MPS value. If the current value of MPS is 0, we calculate the ctxIdx of significant_coeff_flag while that of last_significant_coeff_flag is calculated when MPS is 1.

For other SEs, only the ctxIdx of the first bin depends on the neighboring information and others are related only to the bin index (binIdx). Therefore, we can calculate the ctxIdx of two bins without any stall.

3.4 Neighboring information simplification and pre-fetching

The calculation process for ctxIdx uses the SE information from the left and top MB or MB pairs, which also induces a large overhead in memory access. Furthermore, in the main profile that adopts MBAFF, the SE information of two-line MBs should be stored. For 1080P video, the MB number for one line is 68 and the complete SE information used by a CABAC decoder for one MB is 267 bits. Hence, it costs more than 4 KB to store the neighboring SE information. To reduce the on-chip memory cost, neighboring information should be stored in off-chip memory. However, it is necessary to access neighboring information efficiently to accelerate the decoding of the CABAC decoding engine, while keeping power consumption low. To improve efficiency, two techniques are proposed in our architecture: syntax information simplification and neighboring information pre-fetching.

According to the referred characteristics, we find that it is not necessary to use the complete SE information. Taking MVD as an example, at most 52 bits are required to store the complete information of the MVD for one 4×4 block. In fact, during the calculation of ctxIdx, values greater than 32 are treated as the same. Therefore, only 6 bits are needed to indicate one direction of MVD and the total bit number used to store the information of one 4×4 block is 24. In this case, we can reduce the memory size by half and reduce memory access operations from two to one (provided that the memory data width is 32 bits). By exploiting the simplification technique for all SE information of one MB, the size can be reduced from 267 to 138 bits, which saves almost 50% of memory consumption.

Even with SE information simplification, the decoder efficiency is affected by the latency of the off-chip memory access. Therefore, a second technique of data pre-fetching and buffering is adopted to hide the long latency of off-chip memory access. The pre-fetching technique is proposed based on the fact that for decoding many SEs, only the ctxIdx of the first bin depends on the neighboring information. The others are fixed or related only to the pre-decoding bin. When decoding non-first bins, the neighboring information of the next SE can be pre-fetched from off-chip memory to the on-chip buffer. Similar to the context model buffer, a 64-bit circular buffer is used to store the neighboring SE. However, there are two cases in which the pre-fetching technique does not work: (1) when the current decoding SE contains only one bin; or (2) when the type of the next SE depends on the result of the current SE. In the first case, if the width of the SE is only one bin, it depends mostly on only one bit of the neighboring information. Thus, neighboring information of several SEs can be fetched together and stored in the buffer. In the other case, the prediction is used to decide which SE's neighboring information should be fetched. If the prediction turns out to be wrong, the fetched data will be flushed and the correct neighboring information will be loaded.

4 Hardware implementation of the CABAC decoder

Based on the proposed optimization techniques, the hardware architecture for the CABAC is shown in Fig. 4. The CABAC CTRL module acts as a central controller of the H.264/AVC CABAC decoder. It provides control signals for three circular buffers to offer corresponding information for the decoding engines and manages the decoding flow of the SE. The CABAC CTRL selects the correct neighboring information for the Ctx circular buffer. It will also decide how many decoded bins are valid in this cycle according to the values of sign bit 1 and sign bit 2, which correspond to the comparison results in Fig. 3b. The Ctx circular buffer calculates the context indexes of both the current and next bins on the assumption that the decoding value of the current bin hits MPS. Two pStateIdxs are provided to the decode decision engine (Fig. 4). NSE, Ctx, and BS buffers offer

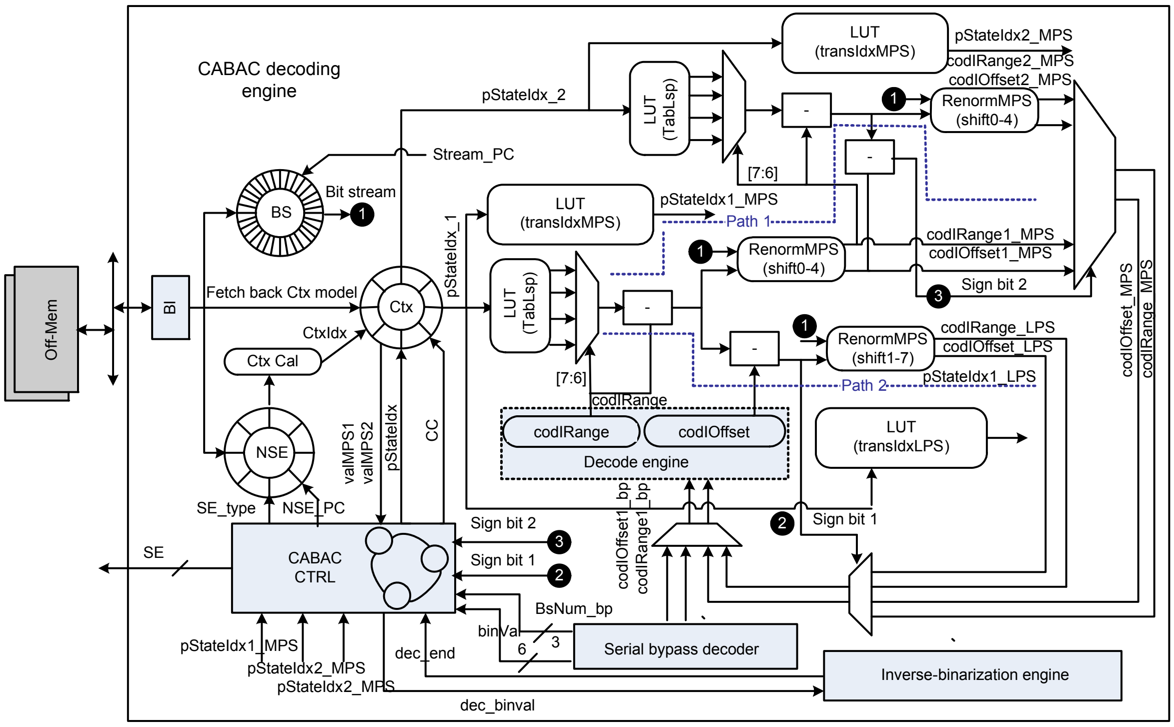

Fig. 4 The proposed context-based adaptive binary arithmetic coding (CABAC) architecture

neighboring SEs, context models, and bit stream, respectively, for the CABAC arithmetic engine. They are all organized as circular buffers, which act as infinitely large buffers and serve as an interface between the off-chip DRAM and the hardware decoder. All buffers share the bus interface (BI) to access DRAM alternately. The inverse-binarization module accepts the decoded bin string and decides whether the decoding process of the current SE has been terminated. There are two decoding engines, the decode bypass engine and the decode decision engine. The decode bypass engine consists of two decoding cores, a head zero detector, and two series of decode bypass engines, which aim to decode different parts of the SEs. The decode decision engine is composed of two concatenated MPS branches and one LPS decoding branch; when the values of two serial bins are both located in the MPS range, it is able to decode two bins in one cycle.

4.1 Circular buffer

During the CABAC decoding, four types of memory access are required: (1) read bit stream; (2) fetch neighboring SE; (3) load context models; (4) update context models. To reduce the on-chip memory size, all of the above information is stored in off-chip memory in the proposed architecture. But if the data is fetched each time, it brings long latency and a large amount of extra power consumption caused by frequent visits to the external memory. A circular buffer is introduced to solve these problems. It serves as a bridge between the off-chip memory and on-chip CABAC decoding. All three buffers are organized in a similar way. The buffer of context models contains at most 112 bits for 16 context models (each context model consists of 7 bits). There are five pointers: context counter (CC) and four water-level markers pointing to each 28-bit position (Fig. 5).

The moving step of the CC is controlled by the CABAC controller model. There are two conditions to increase CC: (1) One SE decoding completes and then CC points to the context model of the next SE; (2) The pointed context model will not be used any more in current SE decoding. Four water-level markers are used to trigger the replacement behavior of context models and they are fixed to four pointers address27, address55, address83, and address111, respectively. The circular buffer contains a self-refilling mechanism. Each time the CC steps over one water-level marker, the refilling process will be carried out: the updated context models in the buffer are written into the memory and new ones are fetched.

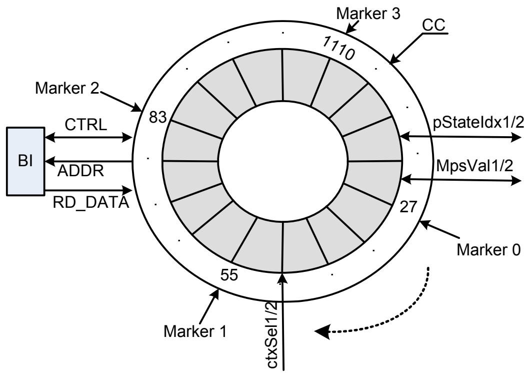

Fig. 5 Circular buffer

Fig. 6 shows the organization architecture of context models in the memory and the buffer refilling process. All context models are divided into four categories according to the slice type and cabac_init_idc value. Each category is organized into 29 groups in the same way, according to the rules given in Section 3.2. In Fig. 6, the SEs, mb_type (I_Slice), intra_chroma_pred_mode, prev_intra4×4_pre_mode_flag and rem_intra4×4_pred_mode_flag are organized into group 3, while code_block_pattern and mb_qp_delta are organized into group 5. If mb_type is intra (provided that it is true in the given example), the SEs listed above are decoded in sequence. After decoding the SE mb_type, the context models with an index of from 3 to 10 are not used any more in this MB level decoding. So they are written back into the memory and the context models of group 5 are loaded to fill the empty place. Fig. 6 shows the refilling steps. In some conditions, the type of the next decoding SE is unknown when the buffer has empty space. Therefore, one key problem that needs to be solved is deciding which group of context modules should be fetched for the context buffer refilling process. When the next group of context modules is unknown, the prediction technique is used to pre-fetch the context models. If the prediction is wrong, the filled buffer will be flushed and correct ones are fetched.

The circular buffers of bit stream and neighboring SE are organized in a similar way, except that they have no write-back logics. For the stream buffer, the bits will not be used any more after its decoding.

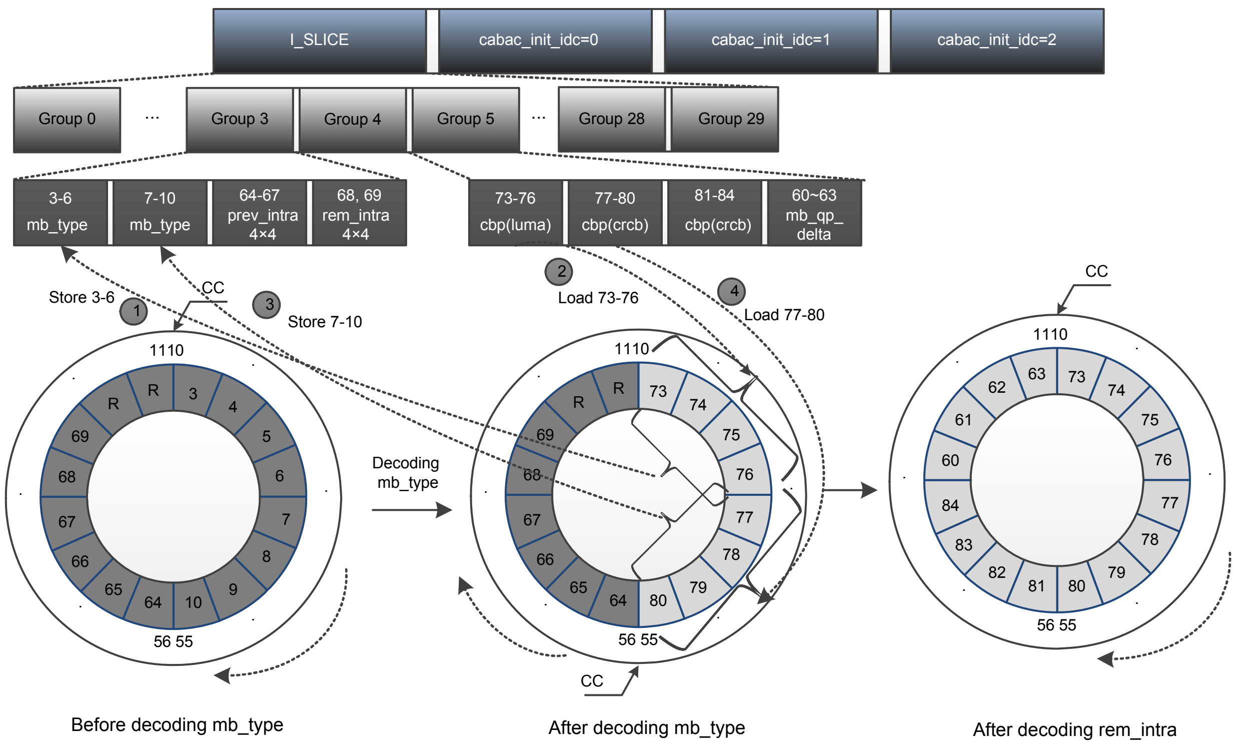

Fig. 6 Architecture of context models in memory and the buffer refilling process

Therefore, the write-back operation is saved. As for the neighboring SE, the writing logic of the current decoding SE is shared with other parts of bit stream parsers. Moreover, the bit stream buffer is 128 bits, which can also be used for CAVLC decoding, similar to the design of Xu *et al*. (2006). But the buffer size of the neighboring SE is only 64 bits. By using the circular buffer structure, performance is maintained as if there is an infinite buffer providing continuous information to the CABAC arithmetic logic.

4.2 Re-normalization engine based on a head-one detector

The last step of the decode decision engine flow is renormalization (Fig. 3). To keep the precision of the whole decoding process, the refined codIOffset and codIRange have to be renormalized to ensure that the codIRange is not less than 256. For example, if the refined codIRange is 9’b000011010, the codIRange should be shifted four bits while the codIOffset reads four bits from the bit stream during the renormalization process. Based on the principle of renormalization, we find that if we locate the first appearing ‘1’ inside the codIRange, we can successfully decide the number of bits of the codIRange to shift and of the codIOffset to read. Moreover, the renormalization process is part of the critical timing path in CABAC hardware decoder implementation. To improve clock frequency, this path must be kept as short as possible. Thus, a parallel ‘head-one detector’ re-normalization architecture is proposed (Fig. 7). Nine bits of the codIRange are split into three parts (3-bit vector), each of which determines whether there is a ‘1’ among three input bits. These results determine which part should be further tested. Compared with previous designs (Yang *et al*., 2006; Yang and Guo, 2009), the re-normalization path is reduced by almost 50%.

4.3 Bypass decoding engine based on a head-zero detector

The bypass decoding engine is used mainly to decode the suffix of two syntax elements, i.e., MVD and coeff_abs_level_minus1, which account for more than 90% of the whole CABAC decoding (Chang and Lin, 2009). Therefore, improving the bypass decoding engine throughput may increase the CABAC decoding efficiency greatly. The bypass mode is relatively simple, since it is not necessary to access

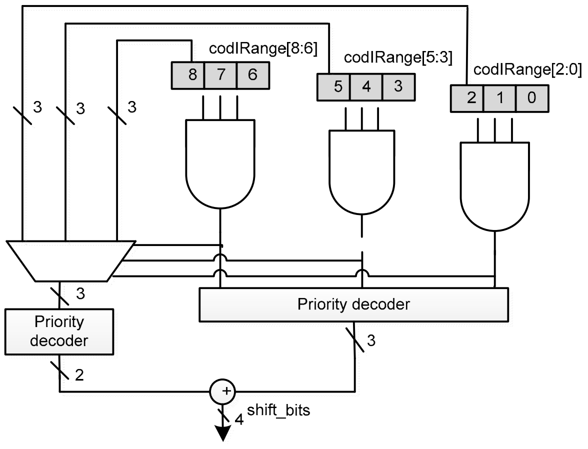

Fig. 7 Re-normalization engine based on a head-one detector

context models, and the codIRange and codIOffset are updated simply. We can merge several engines to decode several continuous bins so that the decoder can process them in parallel to increase data throughput. To improve the parallelism, Yang and Guo (2009) adopted four series of bypass decoding engines to decode four bins in each cycle. However, their method does not balance the critical path with the decision decode engine, and decreases the frequency of the whole decoder. To keep the critical timing path balanced with the proposed decode decision engine, we merge two series of normal bypass engines to decode the bins in parallel.

Both MVD and coeff_abs_level_minus1 use UEBk (kth order Exp-Golomb) binarization process, which has a sequential feature in that the suffix is composed of a series of 1's followed by a '0', and the length of the remaining bins depends on the length of 1's. Furthermore, according to the decode bypass engine mechanism, the bin value '1' must be decoded by the LPS branch of the decoder. Hence, we propose a new bypass decoding architecture that consists of two parts: a head-zero detector with an optimized LPS branch of the bypass decoding engine to locate the first appearance of '0', and then two series of normal decode bypass engines to decode the remaining bins. According to the statistical results shown in Fig. 8, for most video sequences, about 90% of the syntax elements coding with UEBk contain no more than four continuous 1's. Therefore, we propose a head-zero detector to locate '0' in each four bits. Fig. 9b shows the working mechanism of our bypass decoding engine.

In the original bypass decoding mode, the number of decoding cycles is the same as the length of bins; i.e., it takes 15 cycles to decode the binary sequence (Fig. 9a). Shi *et al.* (2008) merged the bypass decoding algorithm to decode two successive bins at the same time, while keeping the critical decoding path short. However, their method does not explore the binarization features of SEs. Our hardware architecture explores the features of UEBk binarization to decode four 1's in one cycle and two bins for the remaining bins. It maintains the bypass decoding critical path balance with the decode decision engine, while increasing the throughput greatly. Fig. 10 shows the architecture of the proposed bypass decoder. A bypass controller module is responsible for

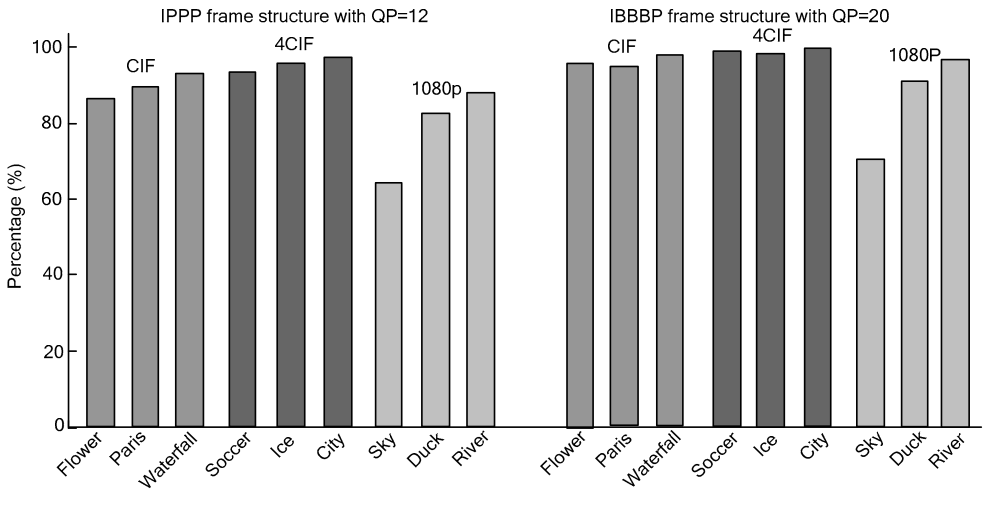

Fig. 8 Percentage of UEBk coding syntax elements with no more than four continuous 1's

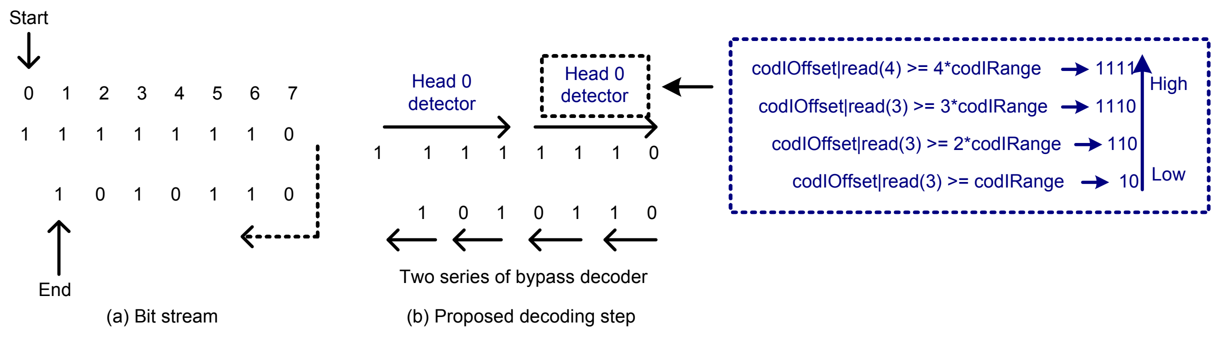
Start
0 1 2 3 4 5 6 7
1 1 1 1 1 1 1 0
1 0 1 0 1 1 0
End
(a) Bit stream
Head 0 detector
Head 0 detector
1 1 1 1 1 1 1 0
1 0 1 0 1 1 0
Two series of bypass decoder
(b) Proposed decoding step
codIOffset|read(4) >= 4*codIRange → 1111
codIOffset|read(3) >= 3*codIRange → 1110
codIOffset|read(3) >= 2*codIRange → 110
codIOffset|read(3) >= codIRange → 10
High
Low

Fig. 9 Working mechanism of the new bypass decoding engine

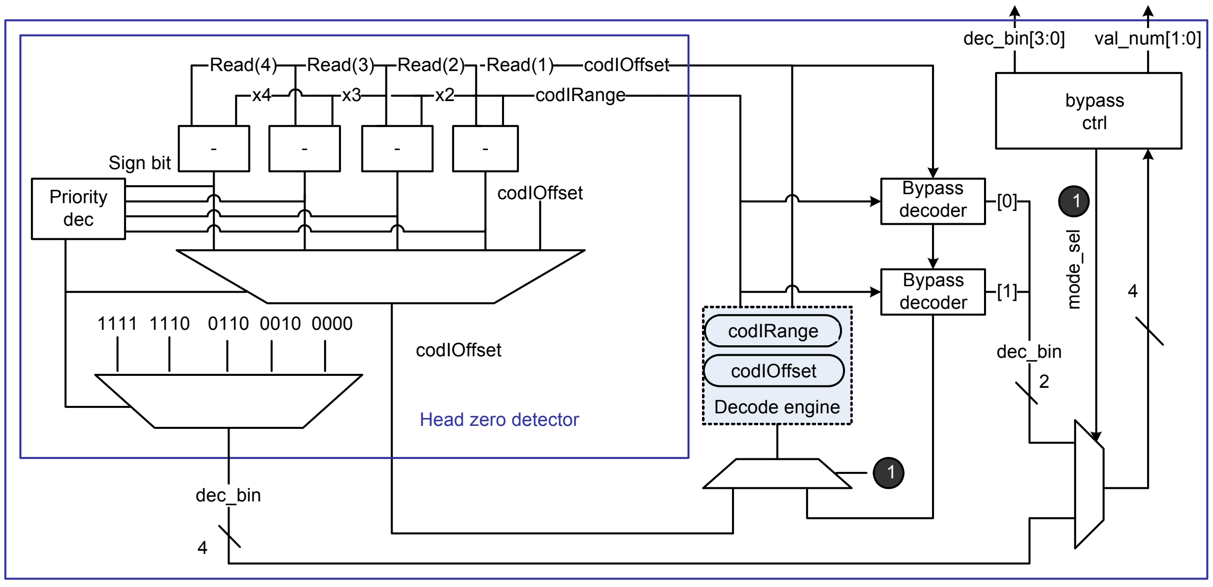

Fig. 10 Architecture of the new bypass decoder

deciding the bypass decoding engine, i.e., a head-zero detector or serial bypass engines. When the decoded bins are sent to the CABAC CTRL module with dec_bins, an extra signal named val_num is set to indicate the real valid bins.

4.4 Decode decision engine

According to the analysis in Section 3.1, a reorganized decode decision engine is proposed with two concatenated MPS branches and one LPS branch. The proposed engine can be treated as one complete normal decode decision engine and one look-ahead decoding MPS branch (Fig. 4). When the first bin is decoded in the normal way, the second bin is decoded in the extra MPS branch on the assumption that the MPS branch is hit for the first bin. Yang and Guo (2009) proposed a similar look-ahead architecture consisting of two normal decode decision engines, but the decoding of two bins in one cycle takes more time than in our design. Also, its critical path is similar to two series of the decision decoder engine. Therefore, the clock frequency of its architecture is restricted. In our design, the critical timing path is almost the same as that of one decode decision engine. There are three different decoding properties in the proposed engines: (1) The LPS branch of the engine is taken; (2) Only the MPS branch of the normal decode decision engine is taken; (3) Both the normal engine and look-ahead decoding MPS branch are taken. The first two properties take path 2 as the critical path, while the last one takes path 1 (Fig. 4). Path 1, which consists of three 9-bit minus operators and one MPS renormalization module, is the actual critical timing path of the proposed engine. When two serial MPS values are selected as the results of decoding bins, two bins can be decided at the same time. Otherwise, only the decoding value from the normal engine is selected.

5 Experimental results

We have implemented the CABAC hardware decoder with these proposed optimization techniques in synthesizable Verilog HDL, and integrated it into the whole H.264 decoder with other parts of our design (Li *et al*., 2010; 2011; Ma *et al*., 2011). Table 2 shows the experimental results of the average number of processing bins in each cycle with different QP and frame structures. All the test sequences generated by H.264/AVC reference software had the 4:2:0 color format and a frame rate of 30 frames/s. The performance results show that the average data processing rate of our design can achieve about 1.5 bins per cycle and the decoder has better performance for 1080P video sequences with high bit-rate coding. Furthermore, the RTL codes of the CABAC decoder were synthesized using a Synopsys Design Compiler with the SMIC18 silicon process library. In the worst case, the maximum frequency of our design can reach 250 MHz. The throughput of the proposed decoder is 375 Mb/s, which is fast enough to support the H.264 Main Profile @L4.2.

Two key optimization strategies, circular buffer and pre-fetching, were adopted to reduce the DRAM access times and accelerate data fetching in our design. Two experiments were added to illustrate the efficiency of these two techniques by counting the DRAM access times and latency. A DRAM model

Table 2 Average data processing rate of the proposed decoder

Picture type	Picture name	QP	Bit rate (Mb/s)			Data processing rate (bin/cycle)		
			IPPP	IBBBP	IIII	IPPP	IBBBP	IIII
CIF	Flower	28	1.73	1.62	5.28	1.52	1.53	1.51
		20	3.25	3.87	8.73	1.49	1.53	1.62
		12	6.39	7.27	13.10	1.51	1.57	1.75
	Paris	28	0.84	0.96	4.03	1.47	1.48	1.47
		20	1.42	2.16	7.46	1.45	1.49	1.53
		12	4.07	5.06	12.80	1.48	1.54	1.64
	Waterfall	28	0.81	0.67	3.75	1.49	1.52	1.48
		20	1.33	1.93	8.30	1.48	1.51	1.51
		12	5.20	6.16	14.60	1.50	1.53	1.59
4-CIF	Soccer	28	2.09	2.58	8.84	1.51	1.52	1.51
		20	9.01	9.58	19.30	1.51	1.51	1.51
		12	25.20	25.50	36.60	1.49	1.52	1.58
	Ice	28	0.89	1.04	3.26	1.49	1.49	1.47
		20	2.81	3.37	8.17	1.48	1.49	1.50
		12	11.80	12.20	20.80	1.47	1.49	1.55
	City	28	2.39	2.68	11.20	1.49	1.52	1.48
		20	10.30	11.50	23.30	1.51	1.52	1.51
		12	28.70	29.70	42.30	1.48	1.51	1.59
1080P	Sky	28	35.9	23.1	34.5	1.55	1.58	1.53
		20	50.8	57.6	71.8	1.59	1.58	1.54
		12	127.0	134.0	152.0	1.59	1.59	1.60
	Duck	28	48.6	45.5	60.9	1.54	1.55	1.55
		20	147.0	148.0	168.0	1.50	1.51	1.51
		12	298.0	297.0	310.0	1.54	1.54	1.56
	River	28	30.1	30.4	33.9	1.58	1.57	1.57
		20	73.7	74.9	84.3	1.52	1.52	1.54
		12	170.0	171.0	185.0	1.54	1.54	1.55

was included in the simulation. It took 10 cycles to access an address not related to the previous address and one cycle delay for burst accessing a continuous address. Table 3 shows the contribution of these strategies to a reduction of about 41% in memory bandwidth and 90% in latency for data fetching. The reduction of memory access times is gained mainly from the context models buffer, which avoids inefficient updating of context models. The pre-fetching technique is able to efficiently hide the data latency. When the type of the next SE depends on the result of the current one, the prediction technique is adopted to pre-fetch the context models or neighboring information. The prediction accuracy of context models and neighboring information is shown in Table 4.

Moreover, by exploiting the feature of the decode decision engine that the MPS branch is much simpler than the LPS branch, our design concatenated two MPS branches while keeping the LPS branch the same. If two series of MPS occur, the new decode decision engine can decode two bins in one cycle. The time ratio in which the engine decoded two bins in one cycle for different QP and frame structures is shown in Fig. 11. In almost 70% of the decoding time, the throughput of the engine was 2 bins per cycle. From the experimental results, the decoding speed is clearly higher for video sequences with higher time ratios for two-bin decoding. We can conclude that the proposed decode decision engine with two concatenated MPS branches is quite efficient.

Table 4 Prediction accuracy of context models and neighboring information pre-fetching[*]

Picture type	Picture name	QP	Hit_rate (%)[**]	
			Ctx	Nei
CIF	Flower	28	91.9	89.1
		20	92.3	89.4
		12	90.4	89.8
	Paris	28	88.6	83.4
		20	89.2	85.1
		12	90.0	87.8
	Waterfall	28	93.5	90.1
		20	89.4	85.5
		12	96.1	93.1
4CIF	Soccer	28	92.6	89.1
		20	93.3	90.0
		12	96.0	92.4
	Ice	28	88.6	83.7
		20	87.9	82.9
		12	89.9	85.1
	City	28	91.4	88.0
		20	95.8	92.3
		12	96.8	92.3
1080P	Sky	28	91.9	92.0
		20	92.5	92.3
		12	97.1	97.8
	Duck	28	94.9	92.8
		20	98.5	94.1
		12	98.7	96.2
	River	28	96.9	96.3
		20	97.6	93.0
		12	98.1	94.1

[*] For IBBP frame structure. [**] hit_rate=(valid pre-fetching times)/(total pre-fetching times)×100%

Table 3 Reduction in DRAM access using the circular buffer[*]

Picture type	Picture name	DRAM access times per frame			Data fetching penalty (cycle/frame)[**]		
		No buffer	With buffer	Reduction (%)	No buffer	With buffer	Reduction (%)
CIF	Flower	28062	16502	41.2	84768	7005	91.7
	Paris	17353	10089	41.9	55903	6324	88.7
	Waterfall	20273	11481	43.4	66830	6827	89.8
4CIF	Soccer	92833	52949	43.0	318544	19845	93.8
	Ice	35176	19928	43.3	107560	14909	86.1
	City	107435	61471	42.8	326359	17215	94.7
1080P	Sky	462700	269012	41.9	1514699	107127	92.9
	Duck	989515	588275	40.5	3141291	66760	97.9
	River	630603	364723	42.2	1973758	62057	96.9

[*] For IBBP frame structure, QP=20. [**] The penalty of data fetching is the cycle cost between the data request and the data acquisition; when the information has been in the circular buffer, the penalty equals zero

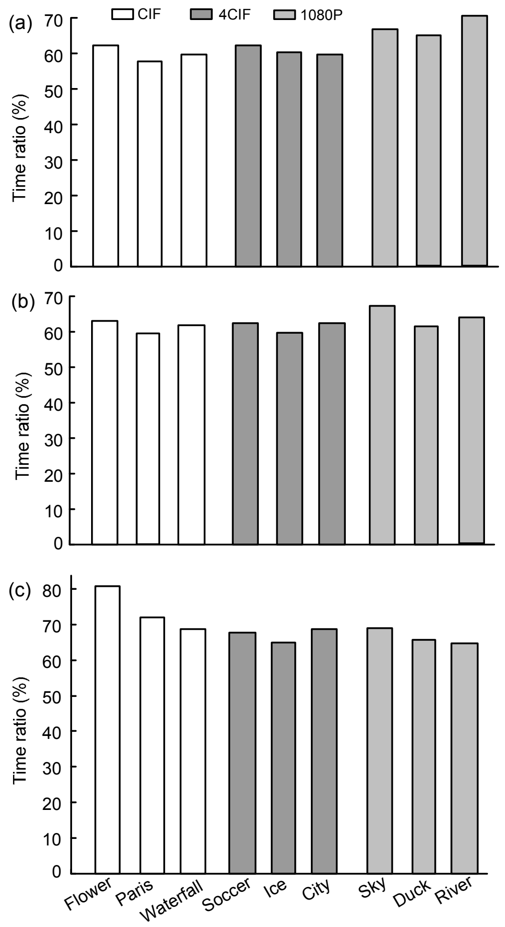

Fig. 11 Time ratios in which the engine decodes two bins at one cycle

(a) IPPP frame structure, QP=28; (b) IBBP frame structure, QP=20; (c) IIII frame structure, QP=12. Time ratio of two-bin decoding=(number of two-bin decoding cycles)/(total number of decoding cycles)×100%

Table 5 shows a comparison of results from our proposed design with those from existing methods. The design of Kuo *et al.* (2011) achieved the lowest hardware cost by optimizing the memory system. However, it applies only to a single-bin engine and sacrifices the utilization of two-bin arithmetic decoding engines for some SEs, and decodes only 0.95 bins per cycle on average. Although the design of Liao *et al.* (2012) achieved the highest throughput, the pipeline-based architecture and on-chip storage of context models increase the hardware cost on gate count greatly. Also, its context model fetching logic is more complicated since there are two memory sources of context models. Compared with previous methods, our design provides high throughput while keeping the area cost low (with the highest throughput per second per thousand gates). By using three circular buffers, our design reduces the on-chip SRAM to zero. The buffers can provide continuous information for the engines in most cases. In addition, unlike the designs of Yang *et al.* (2006) and Yang and Guo (2009), whose critical timing paths are about equal to the length of two decode decision engines or four decode bypass engines, the critical path of our design is quite similar to the length of one decode decision engine. Therefore, in a similar silicon process, the maximum clock frequency of our design is increased by 50% compared with the results of previous studies (Yang *et al.*, 2006; Yang and Guo, 2009). Furthermore, the hardware cost of our design is 28% less than those of Shi *et al.* (2008) and Yuan (2008), whose high hardware overheads are caused mainly by complicated hardware pipelines.

Table 5 Comparison between different designs

CABAC decoder	Frequency (MHz)	Throughput (Mb/s)	Technology	Gate count ($\times10^3$)	Memory type	Memory size (KB)	Throughput (Mb/(s·kgate))[d]
Proposed design	250	375	SMIC18	20.71[a]	N/A	N/A	18.1
Liao *et al.* (2012)	264	446–485	UMC90	51.276	DP+Reg	DP: 0.179; Reg: 0.222	8.6–9.4
Kuo *et al.* (2011)	150	143–239.4[b]	UMC90	16.291[c]	SP	3.36	7.3–12.2
Chang and Lin (2009)	200	60–100	TSMC18	138.226	N/A	N/A	0.43–0.72
Yang and Guo (2009)	140	120.4	TSMC18	34.955	SP	0.43	3.40
Shi *et al.* (2008)	200	254	18 μm	29.0[c]	DP	10.81	6.4
Yuan (2008)	250	160	TSMC18	35.6	SP	0.49	4.5
Chen and Lin (2007)	137	N/A	TSMC13	40.762	N/A	N/A	N/A
Yang *et al.* (2006)	120	253–462	TSMC18	83.157	N/A	N/A	3.1–5.6
Yu and He (2005)	150	N/A	18 μm	30	N/A	0.42	N/A

[a] All of the register-based circular buffers are included in the total gates of the design; [b] In the best case, the maximum throughput will achieve 239.4 Mb/s by increasing the frequency to 249 MHz; [c] The on-chip SRAM is not included in the gate count of the designs; [d] The memory size is considered as part of the total gate size when calculating the throughput per second per thousand gates. SP: single port; DP: dual port

6 Conclusions

We have presented a new reorganized decode decision engine with look-ahead ctxIdx calculation logic to improve performance by decoding two bins simultaneously. In the proposed architecture, two MPS decoding branches are concatenated while keeping the critical timing path as one decode decision engine. Look-ahead logic is adopted to calculate the ctxIdx for decoding the second bin on the assumption that the first bin is the MPS value. Experimental results show that the engine can decode two bins simultaneously in almost 70% of decoding cycles. For the decode bypass engine, four LPS branches are merged to decode the beginning 1's of UEB*k* encoding syntax elements, and two concatenated normal bypass engines are adopted to decode other bins. We have also proposed three techniques to divide the 459 context models into 29 groups, and have used three circular buffers to cache one group of context models, neighboring information, and bit stream, respectively, to reduce the frequency of memory access operations. The proposed CABAC decoder can decode 1.5 bins per cycle and has already achieved a real-time video decoding application for the H.264/AVC main profile at level 4.2.

References

Chang, K.H., Lin, Y.L., 2009. A Very High Throughput Fully Hardwired CABAC Decoder. Int. Symp. on Intelligent Signal Processing and Communication Systems, p.200-203. [doi:10.1109/ISPACS.2009.5383868]

Chen, J.W., Lin, Y.L., 2007. A High-Performance Hardwired CABAC Decoder. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, p.37-40. [doi:10.1109/ICASSP.2007.366166]

ITU-T Recommendation H.264:2003. Advanced Video Coding for Generic Audiovisual Services. Telecommunication Standardization Sector of International Telecommunication Union.

Kuo, M.Y., Li, Y., Lee, C.Y., 2011. An Area-Efficient High-Accuracy Prediction-Based CABAC Decoder Architecture for H.264/AVC. IEEE Int. Symp. on Circuit and Systems, p.160-163. [doi:10.1109/ISCAS.2011.5937974]

Li, C.S., Huang, K., Yan, X.L., Feng, J., Ma, D., Ge, H.T., 2010. A High Efficient Memory Architecture for H.264/AVC Motion Compensation. IEEE Int. Conf. on Application-Specific Architecture and Processors, p.239-245. [doi:10.1109/ASAP.2010.5540963]

Li, C.S., Huang, K., Xiu, S.W., Ma, D., Ge, H.T., Yan, X.L., 2011. High efficient pipeline design and implementation for sub-pixel interpolation process in H.264/AVC. *J. Zhejiang Univ. (Eng. Sci.)*, **45**(7):1187-1193 (in Chinese). [doi:10.3785/j.issn.1008-973X.2011.07.008]

Liao, Y.H., Li, G.L., Chang, T.S., 2012. A highly efficient VLSI architecture for H.264/AVC level 5.1 CABAC decoder. *IEEE Trans. Circ. Syst. Video Technol.*, **22**(2): 272-281. [doi:10.1109/TCSVT.2011.2160752]

Ma, D., Huang, K., Chen, H.F., Yu, M., Yan, X.L., 2011. Mixed increasing filter pipeline design for H.264/AVC deblocking filter. *J. Zhejiang Univ. (Eng. Sci.)*, **45**(7): 1206-1214 (in Chinese). [doi:10.3785/j.issn.1008-973X.2011.07.011]

Shi, B., Zheng, W., Lee, H.S., Li, D.X., Zhang, M., 2008. Pipelined Architecture Design of H.264/AVC CABAC Real-Time Decoding. 4th IEEE Int. Conf. on Circuits and Systems for Communications, p.492-496. [doi:10.1109/ICCSC.2008.110]

Xu, K., Choy, C.S., Chan, C.F., Pun, K.P., 2006. Power-Efficient VLSI Implementation of Bit Stream Parsing in H.264/AVC Decoder. IEEE Int. Symp. on Circuit and Systems, p.984-988. [doi:10.1109/ISCAS.2006.1693839]

Yang, Y.C., Guo, J.I., 2009. High-throughput H.264/AVC high-profile CABAC decoder for HDTV applications. *IEEE Trans. Circ. Syst. Video Technol.*, **19**(9):1395-1399. [doi:10.1109/TCSVT.2009.2020340]

Yang, Y.C., Lin, C.C., Chang, H.C., Su, C.L., Guo, J.I., 2006. A High Throughput VLSI Architecture Design for H.264 Context-Based Adaptive Binary Arithmetic Decoding with Look Ahead Parsing. IEEE Int. Conf. on Multimedia and Expo, p.357-360. [doi:10.1109/ICME.2006.262510]

Yu, W., He, Y., 2005. A high performance CABAC decoding architecture. *IEEE Trans. Consum. Electron.*, **51**(4): 1352-1359. [doi:10.1109/TCE.2005.1561867]

Yuan, T.C., 2008. A Novel Pipeline Architecture for H.264/AVC CABAC Decoder. IEEE Asia Pacific Conf. on Circuit and Systems, p.208-311. [doi:10.1109/APCCAS.2008.4746021]