Variable-Bin-Rate CABAC Engine for H.264/AVC High Definition Real-Time Decoding

Peng Zhang, Don Xie, and Wen Gao, *Member*, *IEEE*

***Abstract*—This paper presents an efficient VLSI architecture for H.264/AVC Content-Adaptive Binary Arithmetic Code (CABAC) decoding. We introduce several new techniques to maximize the parallelism of the decoding process, including variable-bin-rate strategy, multiple-bin arithmetic decoding and efficient probability propagation scheme. The CABAC engine can ensure the real-time decoding for H.264/AVC main profile HD level 4.0. Synthesis results show that the multi-bin decoder can be operated up to 45MHz, and the total logic area is only 42K gates when targeted at TSMC's 0.18um process.**

***Index Terms*—CABAC, parallel architectures, real-time, video coding.**

I. Introductions

Improvements in advanced video coding technology push the consumer video products into the trend of higher quality, more functionality and lower cost, covering more applications such as TV broadcasting, disc storage and wireless video services. H.264 [1] is the newest version of the international video coding standard, which is developed jointly by ITU-T Video Coding Expert Group (VCEG) and ISO/IEC Motion Picture Expert Group (MPEG), and is also referred to as MPEG-4 Advance Video Coding (AVC). Compared with previous video coding standards, H.264 adopts a series of innovative coding tools including variable block size, finer inter/intra prediction resolution, more reference frames, quarter pixel interpolation, directional intra prediction, adaptive in-loop de-blocking filter and so on [2]. Those tools can improve the coding efficiency by up to 3dB over a wide range of bit rates and video resolutions, which means 50% communication bandwidth and storage capacity can be saved at the same visual quality [3]. However, high coding performance comes at the price of high computation complexity. The complexity of H.264 encoder is two times higher than that of MPEG-4 simple profile, and ten times for decoder [3].

Among those novel coding tools, CABAC is a significant revolution [2]-[4]. It can save the bit rate by up to 14% on average at the same video quality, compared with the other entropy coding tool such as Context-based Adaptive Variable Length Coding (CAVLC). CABAC also introduces high system implementation complexity for both encoder and decoder. Researches show that it is extremely arduous for General Purpose Processor (GPP) and Digital Signal Processor (DSP) to perform CABAC real time encoding/decoding [3], [12]. The highly recursive arithmetic coding process and complex data dependency in context adaptation limit the possibility of parallelism in hardware implementation, especially for high-resolution video applications.

Optimized encoder architecture for adaptive binary arithmetic coding is widely discussed [5]-[10]. Unlike the encoding process, arithmetic decoding (AD) does not have the luxury of allowing slice-level parallelism. This makes it more difficult to design high performance CABAC decoding engines. Jian-Wen Chen [11] implemented a full hard-wired CABAC decoder, which could decode the CIF video in real time. Wei Yu [12] improved the decoding engine with several techniques, such as context model register grouping and multiple-bin post processing and so on. Thus Wei's decoder could process videos at D1 resolution in real time. Chung-Hyo Kim [19] introduced the most probable prediction method to decode more bins in one decoding loop. Hendrik Eeckhaut [20] speeded up the table accessing delay by precalculation.

The key strategy of the previous CABAC decoders is all bin-by-bin decoding. Even in Wei's and Kim's multiple-bin decoding scheme, the parallel decoding bin number is small. To fully exploit the possible parallelism of H.264 CABAC, we proposed our multiple-bin decoding scheme, which can decode up to 16 bins in parallel. Constant-bit-rate decoding strategy can guarantee the system real time requirement in the theoretically worst cases. Efficient system level design largely reduces the overall complexity while maintaining the system performance. We make the best of H.264 CABAC characteristics to efficiently design our decoding engine architectures, including multiple-bin arithmetic engine, probability propagation module and update circuits. Implementation result shows that the proposed engine can process H.264 main profile level 4.0 in real time at the price of slight area increase.

In the following, we first review the CABAC algorithm of H.264 in section II, where a thorough analysis on real time requirement is also presented in this section. Section III introduces the key principle of the proposed scheme, including variable-bin-rate strategy and multiple-bin decoding process. Detailed architecture and optimization are described

Manuscript received July 1, 2007; revised November 21, 2007. This work was supported in part by the National High Technology Development 863 program of China under Grant No. 2003AA1Z1290, and by Spreadtrum Communication Inc.

Peng Zhang is with the Institute of Computing Technology, Chinese Academy of Sciences, China and Graduate School of Chinese Academy of Sciences, Beijing, China (e-mail: zhangpeng@jdl.ac.cn)

Don Xie is with the Spreadtrum Communication Inc., Beijing, China (e-mail: don.xie@spreadtrum.com).

Wen Gao is with the Digital Media Institute, Peking University, Beijing, China (e-mail: wgao@jdl.ac.cn).

in section IV. Implementation and experimental results are shown in Section V. And Section VI concludes the paper.

II. CABAC Algorithm

The inherently sequentially organized processes of adaptive arithmetic coding make it difficult to implement high-performance architecture by adopting parallelism or pipeline. To maximize the parallelism, careful analysis of CABAC algorithm is necessary.

A. Context-Based Adaptive Arithmetic Coding

Before H.264 CABAC, hardware-based binary arithmetic coders are developed, such as IBM Q-coder [15] and QM-coder [16], M-coder [14], Z-coder [13], MZ-coder [8] and CABAC [4]. They reduce the complexity by the features as low-complexity, multiplier-free, and table-based status update.

H.264 CABAC consists of three elementary processes: *binarization*, *context modeling*, and *binary arithmetic coding*.

1) In the first step, a given non-binary valued syntax element (SE) is uniquely mapped to a binary sequence, a so-called *bin* string. Each bin can be either 0 or 1. To make our statement clear, we call one output bit of arithmetic decoder as a *bin*, and one input bit as a *bit*. Binarization is an important pre-processing step to reduce statistically the alphabet size of SE to be encoded. CABAC adopts an efficient binarization scheme which can significantly reduce implementation complexity. On one hand, most probable symbols are represented by short bin strings to minimize bin decoding processes; and on the other hand, CABAC (almost) relies on a few table-free bin string types: unary code, truncated unary code, k-th order Exp-Golomb code, and fixed-length code.

2) One of the most important properties of arithmetic coding is the possibility to utilize a clean interface between modeling and coding. H.264 CABAC defines 399 context models to track the conditional probability of different bin of different SEs under the condition of certain context information. Each probability is represented by a 6-bit status. And the most probable value of each context bin is also stored in context model. Probability status is updated using fixed probability transition table transIdxLPS and transIdxMPS described in [1] and [4]. If the probability status transits to zero and the next bin to be coded is not equal to the stored most probable value, the most probable value is changed to its reversion (0 to 1 or 1 to 0).

3) Binary arithmetic coding is based on the principle of recursive interval subdivision that involves the multiplication operation (1). When we process one bin, the given interval is divided into two sub-intervals which are associated with least probable symbol (LPS) and most probable symbol (MPS) respectively. The LPS sub-interval is derived by:

$$R_{LPS} = R \cdot p_{LPS} \tag{1}$$

and the MPS sub-interval is R_{MPS}=R-R_{LPS}, where R is the original interval size and p_{LPS} is the probability of LPS. Each bin goes through one interval division process and the binary representation of the final interval constitutes the coded bit-stream. CABAC replaces the complex multiplication (1) with a look-up table rangeTabLPS [1].

We put emphasis on the CABAC decoding process in the following. One H.264 syntax element is presented as a string of bins. Different bins may have different context information. Context information of all the syntax elements is stored in the context table which is indexed by context index (ctxIdx). For a certain bin whose index in the bin string is binIdx, its ctxIdx is determined by the binIdx, the syntax type and neighboring decoded information. Context table contains two kinds of information used for arithmetic decoding. The first one is the probability status (pStatus) which is a uniformed integer to indicate the probability of the case that the decoding bin is MPS. The second one is MPS value (valMPS) which is a binary value to indicate whether the MPS is 1 or 0.

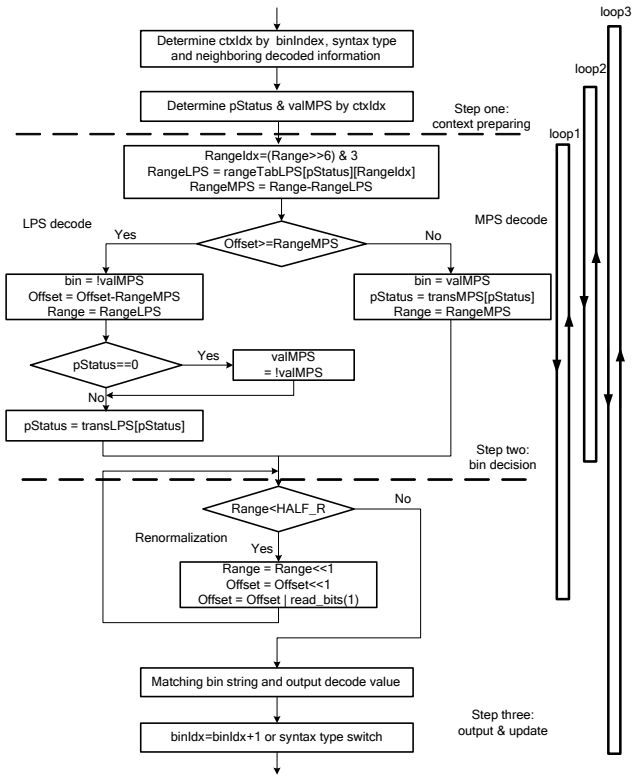

Fig. 1. Bin decoding process

Fig. 1 shows the complete decoding process for one bin. The decoding process has three steps. In step one, context information is prepared as the top two boxes in Fig. 1. The pStatus and valMPS are fetched from context table. In step two, arithmetic decoding is performed for one bin. Arithmetic decoder has two internal status variables. The one is Range which indicates the current interval width as R in (1). Range is split into RangeLPS and RangeMPS by rangeTabLPS and pStatus. The other is Offset which is determined by input bits, and the comparison of Offset and RangeMPS determines the output bin value. If Offset falls into the LPS sub-interval (Offset >= RangeMPS), one LPS is decoded, new Range is set to RangeLPS, and pStatus of current context model is updated

using table transLPS. Otherwise (Offset < RangeMPS), one MPS is decoded, new Range is set to RangeMPS, and pStatus of current context model is updated using table transIdxMPS.

Renormalization and output generation is processed in step three. In renormalization stage, the Range and Offset perform left-shift operation(s) until Range exceeds half of the total interval width (HALF_R). New input bits are read as the lowest bits of Offset during renormalization. The input bit count is equal to the number of bits that Range and Offset have shifted. For example, in Fig. 2(a), $Offset_0$ is less than RangeMPS, a MPS bin is decoded. Offset is unchanged, assuming that $Range_1$ is larger than HALF_R, and no renormalization is needed here. In Fig. 2(b), $Offset_0$ is larger than RangeMPS, so a LPS bin is decoded. And because RangeLPS is less than the HALF_R, Range and Offset are renormalized. RangeLPS and the $Offset_0$ are scaled up by 2 times. The proportion of the Offset in RangeLPS interval is almost kept, except for the lowest bit which is read from input bit-stream.

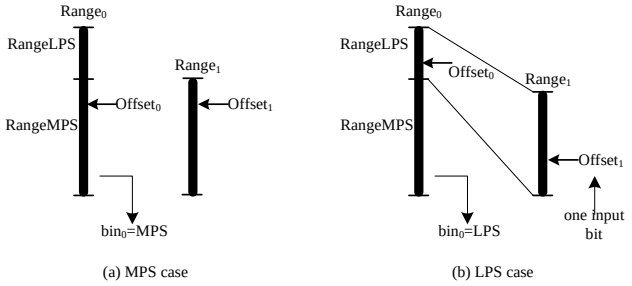

Fig. 2. Bin decision and renormalization

The decoded bin is concatenated with the previous decoded bin string. If the newly generated bin string matches one of the binarization patterns of currently decoding SE, the decoding process of current SE is finished; otherwise binIdx is increased by one and more bins are decoded for the current SE.

B. Data Dependency Analysis

As described in Section II.A, three major data dependencies are extracted as follows:

- Renormalization is dependent on range update.
- Probability transition is dependent on bin decision
- Context switching is dependent on decoded bin

These three data dependency relations lead to three recursive computation loops, which can hardly be sped up by pipelining, and thus largely limit the system performance.

Looking more closely, we can find that there are three directional loops in Fig. 1 and Fig. 3. The loop1 is located in arithmetic decoder engine. Table rangeTabLPS generates RangeLPS, subtractor generates RangeMPS, comparator generates bin decision, and then renormalization generates new range. The delay of the comparator can be hidden into renormalization, if we renormalize both RangeLPS and RangeMPS as divided range and select the two groups of range/offset pairs with bin decision result. And the delay of subtractor can also be reduced when using the rangeTabMPS table to generate RangeMPS directly from pStatus and range. Renormalization can be implemented by an efficient parallel leading-one detection circuit and a barrel shifter. But the table accesses and the renormalization process can not be done concurrently, and they are both time-consuming calculations.

The loop2 is the probability status update loop. The new values of pStatus for both LPS and MPS are generated by looking up transIdxLPS and transIdxMPS simultaneously. The final pStatus is determined by bin decision result. To significantly reduce the effect of access latency of context model RAM, pre-fetching cache [9] and register grouping [12] are proposed in CABAC designs.

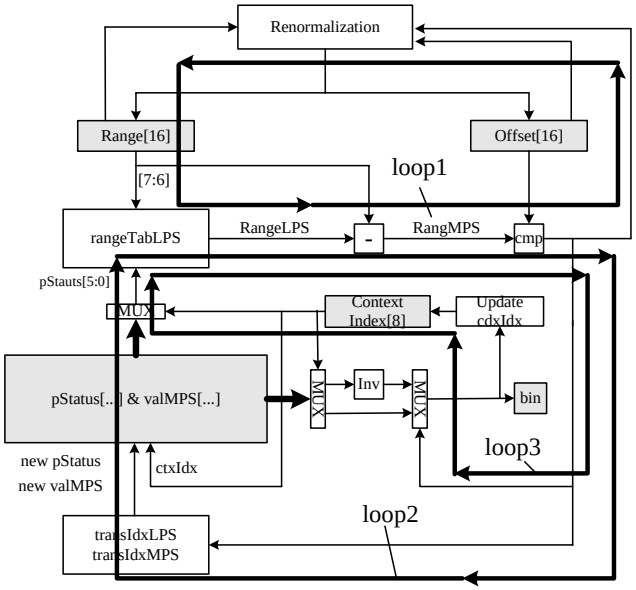

Fig. 3. Data dependency graph of CABAC

The last and most complex loop3 is context updating loop. There are totally 399 context models, and the contexts of successive bins vary according to many factors such as current SE type, neighboring decoded SE value, bin index and binarization pattern. Although most syntax elements are binarized by four basic codes, it is still difficult to perform both SE boundary detection and SE switching in one clock period.

The three iterative computation loops prohibit the deep pipeline scheme from achieving high system performance. Look-ahead computation is a basic method to speed up a recursive FSM [17] [18]. But the speed-up factor is limited by rapidly increased complexity. Multiple-bin arithmetic decoding (MBAD) schemes are adopted in previous works [10] [12], but the performance is limited to at most 3 bins per cycle, and can not meet the requirement of HD applications. In our paper, variable-bin-rate multiple-bin decoding scheme is proposed which can effectively break the three iterative loops at bin level.

C. Useful Characteristics of CABAC

In order to efficiently de-couple the dependency, we made several useful observations on H.264/AVC CABAC:

1) One LPS bin or bypass bin consumes at least one input

bit. The RangeLPS is always less than 0x100 for all context models. So if one LPS bin is decoded, new range needs at least one left-shifting operation to return to the working interval [0x100, 0x1FF]. One bypass decoding consumes one bit as well. Observation 1) is very important for our scheme. We can assume that all the internal (except the last) bins decoded by one input bit are MPSs.

2) One coefficient level (the most frequently occurred syntax element) consumes at least one input bit. Coefficient levels include two SEs, abs_level and sign_flag. The abs_levels are separated by bypass-coded sign_flags. Due to 1), bit consumption occurs at sign_flag. In addition, suffix bins of concatenated unary/k-th order Exp-Golomb (UEGk) are bypass-coded, and this limits the maximum number of bins of one coefficient level to sixteen in one clock cycle.

3) Overwhelming majority of coded bits are assigned to three kinds of SEs (referred to as MB data SE): intra prediction modes, motion information and residual information (up to 95% of total bits on average). This gives us a hint to implement these SE decoding by separating them from other macroblock (MB) SEs (referred to as header SEs).

4) As for context models, some syntax elements (referred to as category-one) use independent context models for successive bins. Category-one SEs include significance map and reference index. Separate context models are accessed when decoding consecutive bins. Newly updated pStatus will not be used when decoding one input bit. So the probability update processes can be done together after one bit is decoded.

5) Some other syntax elements (referred to as category-two) use the same context model for a series of bins, such as coefficient level, motion vector difference and intra prediction modes. No context switching is needed for these bins, but the pStatus of consecutive bins are derived by traversing a series of status transition tables (transMPS).

6) The requirement of decoding speed can be derived by several aspects. But the bin-rate constraint is much stricter than bit-rate constraints. So if we can decode one input bit per cycle, the required system frequency can be reduced significantly compared with constant-bin-rate schemes. The following sub-section analyzes the requirement in details.

D. Real-Time Decoding Requirement

H.264 defines a series of profiles and levels. Profile defines the coding tool set that a decoder must support, and level defines the resolution & bit rate. Maximum bit rates are defined in levels for different video resolutions, compression ratios, etc. High definition level 4.0 limits the maximum bit rate to 20Mbps, which is enough for most HD compressed videos. But arithmetic coding allows assigning fractional bit to a certain SE bin, so that one input bit can be theoretically unfolded into unlimited bins after decoding. To make decoder implementable, H.264 limits the maximum bin number of one Network Abstraction Layer (NAL) unit to

$$(32/3)\times NumBytesInNAL + PicSizeInPel/32 \quad (2)$$

where NumBytesInNAL is the size of one coded NAL unit in bytes, and PicSizeInPel is the total pixel count in one frame. But there are still two contradictions for designing constant-bin-rate CABAC decoder using the bin count limit in (2):

1. H.264 does not give the maximum bit count for specific NAL units explicitly. We can derive the maximum bit count for one picture by the minimum compression ratio and maximum frame size defined for the profile and level. For high definition level 4.0, the required decoding speed is as high as 268Mbin/s, which is almost impossible for the implementation of the three iteration loops with 0.18μm technology.

2. The bit count limit above is over-estimated because the bit-rate of level 4.0 is limited to 20Mbps. If we sum up (2) of all NALs in one period, Hypothetical reference decoder (HRD) model promises that bit rate fluctuation will not lead to decoder input buffer overflow The sum of first parts of (2) is bounded by bit-rate. Theoretically, small amount of bits may be unfolded into a large amount of NALs. Unlimited NAL number causes that the sum of the second parts of (2) is not predictable.

Finally we summarize that the required processing speed for constant-bin-rate decoding is about 13 times (268M/20M) more than that of constant-bit-rate decoding to guarantee the hard real-time decoding in the worst cases for H.264 HD level 4.0.

III. Overview of the Proposed Scheme

The main idea of our proposed decoding scheme can be summarized as variable-bin-rate strategy and multiple-bin arithmetic decoding.

A. Variable-Bin-Rate Strategy

As we have discussed in section II-D, it is very difficult to decode HD videos with constant-bin-rate scheme. Thus we propose our variable-bin-rate scheme. SEs are separated into two sets: the first one is MB data SEs, such as intra prediction modes, motion vector and reference index, and residual information; and the second one is header SEs, including slice level flags, mb_type, sub_block_type, etc.

For the first set, it makes up the overwhelming majority of the input coded bits and also contributes to the major part of computation complexity. And meanwhile context model switch pattern is relatively simple for these SEs. So we use constant-bit-rate scheme to speed up the MB data SEs.

For the second set, computational workload is not very high, and header parsing is a highly irregular process especially in SE switching and SE boundary detection. So we adopt traditional constant-bin-rate scheme to reduce system complexity while also maintaining real-time processing.

In our constant-bit-rate scheme, the proposed decoder processes one input bit in one clock cycle. All the decoded bins corresponding to the input bit are output in parallel. The number of bins output in one cycle varies from 0 to 16, according to the current range/offset pair and the successive probability status. If the initial range of current bit is less than HALF_R, AD engine reads one more input bit to the offset

value and doubles the range value, with no bins outputting in this cycle. The main advantage of constant-bit-rate scheme is that the renormalization process is saved, since one and only one bit is consumed in one cycle.

The bin overflow problem exists in the case that we can not fully consume one bit in one cycle when the bit contains more than 16 bins. According to the observations in section II-C, the maximum bin number for coefficient level information is 16, so the overflow problem will not occur in coefficient level decoding. For intra prediction modes, motion vectors and reference indexes, there are no bypass coded bins for bit boundary within successive SEs, and hence bin overflow problem may exist. Our decoder generates a termination signal when one SE is fully decoded (binarization pattern is matched). Because each of the mentioned three SEs contains no more than 16 bins, termination signals of the trailing bins are certainly to be active. The bin overflow problem can be solved because we don't read new input bit when one of the termination signals of valid decoded bins is active in the clock cycle. These cycles are referred to as *switching cycles*. Another kind of switching cycle occurs when decoding significant maps, because there are totally 32 bins for one block at most. As we will show below, the effect of switching cycles to system performance is acceptable.

For the header SEs, we adopt the traditional one-bin-per-cycle method, but the working frequency is the same as constant-bit-rate scheme which is relatively easy to achieve. Constant-bin-rate scheme can not certainly guarantee consuming one bit per cycle in all cases. The maximum cycles cost in one SE is the maximum bin number, and these cycles are referred to as *redundant cycles*. Redundant cycles don't affect system performance much either.

B. Multiple-Bin Arithmetic Decoding

The basic principle of arithmetic coding enables non-integer bits to be assigned to SE bins, so one bit may be potentially unfolded into multiple bins. The three computation loops described in section II-B limit the possibility of accelerating multiple-bin decoding with parallel and pipelined schemes. However, by making use of the combination of the observations in section II-C and constant-bit-rate scheme, the three computation loops can be totally broken into bin level, and hence most operations in the three loops are parallelized.

As described, one LPS bin or bypass bin consumes at least one input bit, so the bin value tree of all possible 16 output bins within one bit is depicted as below (Fig. 4).

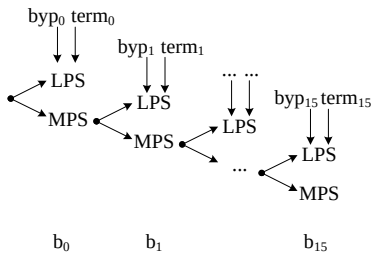

Fig. 4. Output transition tree of one input bit

In the tree of Fig. 4, each dotted node represents one arithmetic decoding unit of one bin, and each distinct path from the root to one of internal nodes or the leaves stands for a string of output bins generated by current bit. There are four cases that may terminate the traversing in the tree:

1) Offset value is larger than RangeMPS. An LPS occurs in this case, current bit is consumed. The output bin string is a single LPS or a series of MPSs followed by one LPS.

2) Range value is less than HALF_R. Range reduction causes the bit consumption. The output bin string is a series of MPS.

3) Current bin is bypass coded (indicated by byp_i). Current bit is consumed. The output bin string is a series of MPS followed by one bypass value (whether offset is larger than the half of range).

4) Current bin is a termination bin (flagged by $term_i$). No new bit is read, and no shifting is done when updating range and offset value. The output bin string is a series of MPSs.

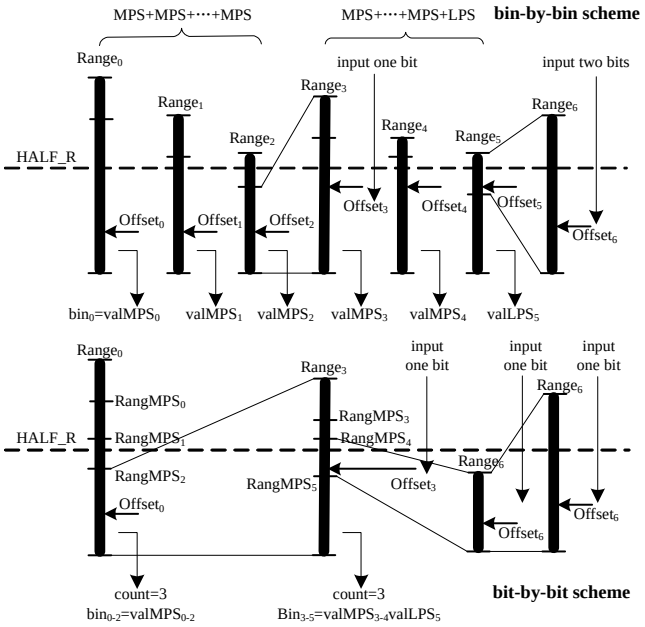

Fig. 5. Comparison between bin-by-bin and bit-by-bit scheme

Fig. 5 demonstrates case 2) and case 1) described above. In bin-by-bin scheme, the first three MPS bins are decoded serially and case 2) occurs. $RangeMPS_2$ is less than HALF_R, and renormalization is performed and one input bit is consumed. As we can see in Fig. 5, when a series of MPS occur, Offset will be unchanged and Range is updated by each RangeMPS serially. In our multiple-bin scheme, a series of RangeMPS (up to 16) are calculated in one cycle, and compared with HALF_R and Offset in parallel. In case 2), the largest one of RangeMPSs which are less than HALF_R is selected as new Range to perform renormalization. The selection also determines the valid decoded bin count. The successive three bins are two MPSs and one LPS, and case 1) occurs. In case 1), the LPS interval [$RangeMPS_{i+1}$,

RangeMPS$_i$] in which the Offset falls is selected as new Range to perform renormalization. Our multiple-bin scheme consumes only one bit per cycle. Two cycles are used to renormalize the Range to be larger than HALF_R as Fig. 5. Case 3) and case 4) are similar with case 2).

In our scheme, when one certain bin is to be decoded, all previous decoded bins corresponding to the bit are assumed to be MPSs, thus the context switching and probability update can be directly processed without full decoding. As to the first computation loop (in section II-B), renormalization process is omitted, and RangeMPS is set directly to range value used by the following bin. As to the second loop, probability status can be derived by iteratively traversing the transIdxMPS table, and the table can be replaced by a saturated incrementer unit. As to the third loop, multiple context indexes for different bins can be derived totally independent of the AD process. The three computation loops are broken by the MPS assumption.

Compared with the constant-rate 16-bin decoding, constant bit-rate decoding can have a significant low complexity. The output transition tree of full constant-rate 16-bin is a complete binary tree which contains as many as 65535 AD units. In these AD units, renormalization can not be saved. Our constant-bit-rate scheme decodes one bit at one clock, the three computation loops is moved to bit level (not bin level). Bit level loop is much looser than bin level loops in computation complexity, because firstly frequency requirement of these bit loops is significantly lower, and secondly computation latency of each AD unit is also largely reduced.

C. System Level Architecture

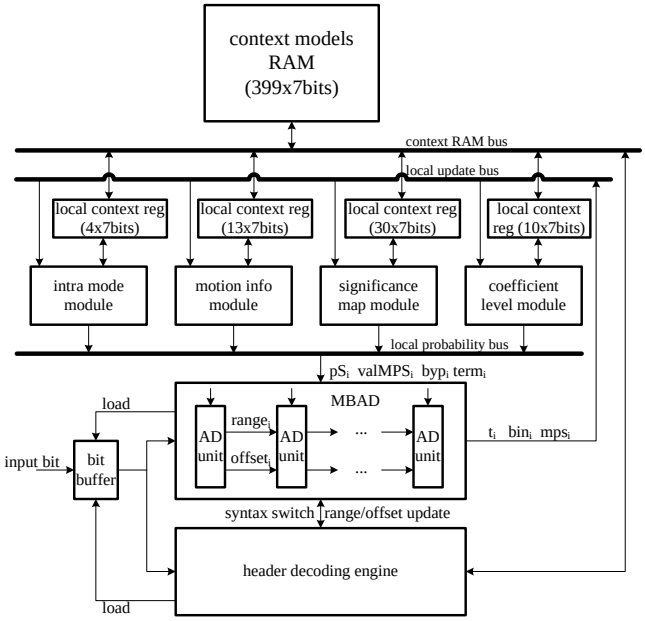

Fig. 6. System level architecture

Fig. 6 shows our system architecture to support variable-bin-rate CABAC decoding. For the context model management, we adopt the two-level memory architecture as [12]. Total 399 context models are stored in on-chip SRAM, with the size of 399x7 bits (six bit for probability status and one for MPS value). Local context register groups (LCRG) are used to speed up the accessing of the context model. All context information which has the possibility to be used in current block is preloaded into local context registers from context model RAM. After decoding one bit, some context registers may be updated by probability propagation (PP) modules, but these new changes will not be written back to context model RAM until different context models are needed to be loaded into local context registers for next block.

Four dedicated PP modules for probability propagation of intra modes, motion information, significant map, and coefficient levels are shown in Fig. 6. These four modules generate up to 16 probability status (pS$_i$) and MPS values (valMPS$_i$) of successive bins. Two major tasks in these modules are SE switching and probability update. Dedicated logic optimization is done for accelerating SE switching process when MPSs occur. And new probability of the same context model is generated by looking up the transIdxMPS table (replaced by saturated incrementer unit). The PP modules also generate bypass flag (byp$_i$) and terminal signal flag (term$_i$) for MBAD engine.

Four PP modules share one MBAD by local probability bus. MBAD performs 16 bin decoding process serially. The update for offset and range is done in MBAD bin by bin. And output updating information is also generated progressively. They are valid flag (t$_i$) to indicate whether the result of current bin is valid (the valid bins constitute the bin string decoded by current bit), output bin value (bin$_i$), and a flag bit (mps$_i$) to indicate whether current bin is MPS. The updating information is sent to local update bus, and PP modules use them to update initial SE and initial context information for next bit decoding.

All header information decoding is processed in header decoding engine (HDE) with context model RAM. Traditional constant-bin-rate decoding scheme is adopted in HDE as [11] [12] and Fig. 3. The values of range and offset in HDE and MBAD are swapped when SE switching between the two engines.

D. System Performance Analysis

Previous CABAC decoders have not given a quantitative performance analysis for real-time decoding, and most of them conclude their real-time guarantee by simulation result. As performance requirements vary a lot for different video scenes, we give a quantitative analysis in the worst cases for our variable-bin-rate scheme.

H.264 level 4.0 limits the bit-rate to 20Mbps. If our MBAD engine can fully decode one bit per cycle, the required working frequency is only 20MHz. But as described in section III-A, we can not decode one bit in every cycle due to redundant cycles and switching cycles. In the worst case, we assume that all the syntax elements that are possible to induce redundant cycles or switching cycles do not consume any bit

at all. The total additional frequency required can be calculated by summing up the total redundant cycles and switching cycles needed per MB and multiplying the sum with MB rate. Header information such as mb_type and sub_partition, is decoded one bin per cycle, so the redundant cycles are counted as the maximum bin count of the corresponding SE. Intra modes, motion information and significant map are bounded by termination signals block by block. Hence at most 16 switching cycles are consumed by one type of SEs in one MB. In a word, due to the redundant cycles and switching cycles, the required system frequency increases from 20MHz to 42MHz. But as we will see below, the frequency requirement of the three bit-level computation loops is easy to meet.

TABLE I
SYSTEM FREQUENCY REQUIREMENT

Syntax Element	Redundant cycles / MB	Switching cycles / MB	Frequency required (MHz)
mb_type	7	-	7x0.24=1.7
sub_patition	6x4	-	24x0.24=6 (inter)
mb_qp_data	2	-	2x0.24=0.5
coded_block_pattern	18	-	18x0.24=4.5
other headers	4	-	4x0.24=1
Switch in intra mode	-	16	16x0.24=4 (intra)
Switch in ref_idx	-	16	16x0.24=4 (inter)
Switch in mvd	-	16	16x0.24=4 (inter)
Switch in sig map	-	2	2x0.24=0.5
constant bit decoding	-	-	20 (level 4.0)
Total			**42(inter), 32(intra)**

The maximum MB rate is 0.24M/s as defined in H.264 level 4.0

IV. DETAILED ARCHITECTURE DESCRIPTION

In this section, we discuss our proposed architecture in detail. We have grouped the data SEs into two categories in section II-C. Significance map and coefficient level are the most typical representatives of the two categories respectively. And in addition, these two SEs constitute the most part of input bit rate. We put emphasis on these two PP modules. Other PP modules such as motion information and intra prediction mode, are similar to them. Detailed architecture of MBAD is also introduced here.

A. Probability Propagation of Significance Map

Significance map consists of two types of SEs, significance flag (SF) and last significance flag (LF). SF indicates whether the coefficient is non-zero, and LF means whether there is no more non-zero coefficient in current block. Each SF or LF is binarized as one bin. The context indexes of SF and LF bins are determined by the coefficient position in the block (0~14).

TABLE II
VARIABLE TRANSITION RELATIONS IN SIGNIFICANCE MAP

Current s_i	Decoded bin b_i	Next s_{i+1}	Next p_{i+1}
0	0	1	p_i+1
0	1	1	p_i+1
1	0	1	p_i+1
1	1	0	p_i

As category-one SE, successive bins of significance map use different context models. Thus the context switching is the main task for significance map PP. We define two iteration variables: s_i (indicates whether current bin is SF, 1 for SF and 0 for LF), and p_i (indicates the coefficient position of current bin). The transition relation can be summarized in Table II.

If current SE is SF and the current decoded bin value is 1 (significant), next SE is LF; otherwise next SE is SF, and coefficient position increases by one. If current SE is LF, next SE is SF and coefficient position increases by one. Table II can be reduced to the following formulas according to the truth table (TABLE II):

$$\begin{aligned} s_{i+1} &= \sim (s_i \cdot b_i) \\ p_{i+1} &= p_i + s_{i+1} \end{aligned} \tag{3}$$

Only one NAND and one 4-1 bit adder operations are needed for one stage of iterator variable propagation.

Concatenating the 16 stages, we get the architecture of significance map PP module (Fig. 7). MPS values (m_i) and pStatus (pS_i) are selected from local context registers with s_i and p_i. In our constant-bit-rate scheme, MPS value is used as decoded bins in each bin stage. New SE flag and new position is updated by (3).

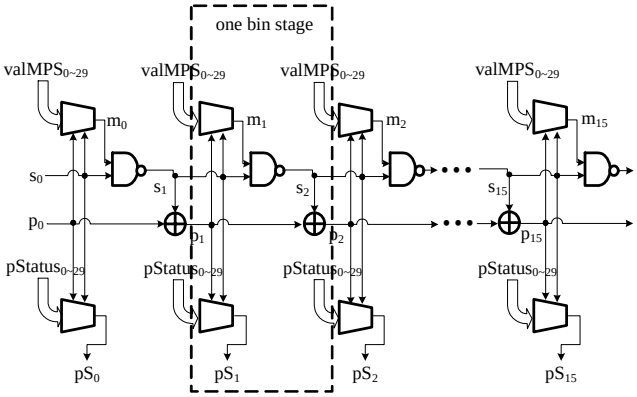

Fig. 7. Architecture of PP for significance map

B. Probability Propagation of Coefficient Level

As described in section II-C, coefficient level consists of two SEs: UEG0-binarized abs_level and bypass-coded sign (Fig. 8). UEG0 binarization consists of a TU-coded prefix with maximum length 14 and an EG0-coded suffix [2].

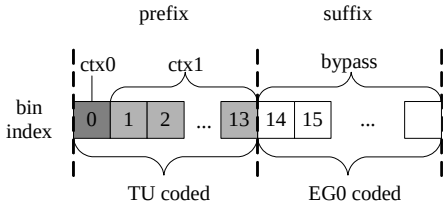

Fig. 8. Binarization of absolute value of coefficient level

The context indexes of abs_levels are adaptively updated with previously decoded levels. Two context models are used

in one coefficient level, one for the first bin (pSb_0) and one for 2^{nd}~14^{th} bins (pSb_1); suffix bins are bypass-coded. The pSb_0 is determined by numDecAbsLevelGt1 and numDecAbsLevelEq1, and pSb_1 is determined by numDecAbsLevelGt1 only.

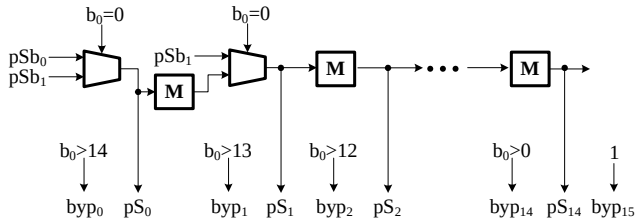

Fig. 9. Architecture of PP for coefficient levels

We use b_i to indicate the index of the bin decoded in the i^{th} bin stage. Due to termination signal of significance map and bypass-coded sign_flag, the first bin of one abs_level must be located in the first bin stage. According to the initial index b_0, pS_i is generated by traversing the transIdxMPS tables (M tables in Fig. 9). And bypass flag byp_i is also generated by the corresponding index. After decoding one coefficient, pS_0 and pS_1 are updated by context models in the local context registers.

C. Multiple-bin Arithmetic Decoding

MBAD engine can produce up to 16 bins until one bit is consumed or a termination signal is asserted. The probability statuses of the 16 bins (pS_i) are generated by PP modules. R_0 and $Offset_0$ are the initial range and offset, which are updated in the previous cycle. Just as PP modules, MBAD engine also consists of 16 bin stages, each of which processes one bin arithmetic decoding. The inputs of one bin stage are current value of range and offset (R_i and $Offset_i$), the probability status of pS_i, bypass flag (byp_i) and termination signal ($term_i$) with i indicating the index of current bin stage. The outputs of one arithmetic decoding stage are valid signal (t_i), MPS/LPS selection signal (mps_i), and decoded bin value (bin_i).

If t_i is equal to zero, the decoded results of current bin stage and all the following bin stages are not valid. That is to say, these output bins will not be appended to the output bin string and these update information will not be used by either R_0/$Offset_0$ update or probability update. The t_i flags are inactive when

(1) LPS occurs. In this case, the sign (highest) bit of the result of subtracting RangeMPS from $Offset_i$ is zero.

(2) Range needs renormalization. In this case, the highest bit of RangeMPS is zero.

(3) The bypass flag or termination signal is active.

(4) The valid flag of previous bin stage (t_{i-1}) is inactive.

The RangeLPS value of each bin stage is generated by table looking up with R_i and pS_i. And RangeMPS is generated by subtracting RangeLPS from R_i. Because all hypothesized decoded bins are MPSs, range value for next bin stage is just associated with RangeMPS value of current bin stage. Meanwhile, Offset value for next bin stage is equal to the result of subtracting RangeMPS from $Offset_i$. According to the arithmetic decoding process described in Fig. 1, if $Offset_i$ is less than or equal to RangeMPS, the actual decoded bin is LPS; otherwise, the decoded bin is MPS. So we reuse the sign bit of the result of subtracting RangeMPS from $Offset_i$ as mps_i.

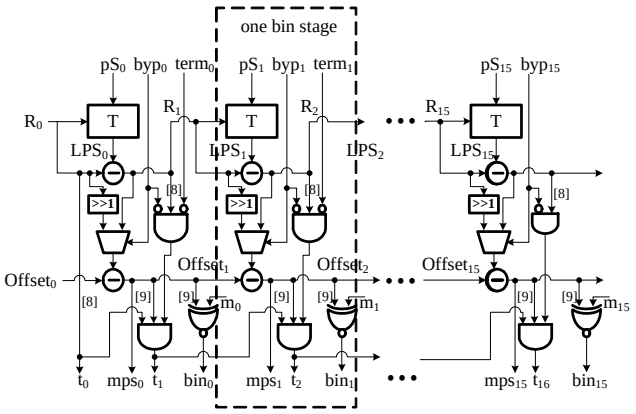

Fig. 10. Multiple-bin arithmetic decoding architecture

Output bin is generated by the MPS/LPS decision signal mps_i and MPS value (m_i). Because bin_i is equal to m_i when mps_i is high, or bin_i is set to be ~m_i when mps_i is low, we can efficiently implement the logic with an XNOR operation.

Decoding functions of bypass-coded bin and termination bins are also seamlessly integrated into our MBAD architecture. When bypass-coded bin occurs, the threshold value of $Offset_i$ for making MPS/LPS judgment changes from RangeMPS to R_i>>1. As we can see in Fig. 10, propagation of range value is broken by the bypass cases. When bypass bin occurs, the following bin stages are bound to be invalid (t_i is equal to 0), so the bypass MUX is omitted in range propagation path, and this shortens the range propagation delay. Termination signal directly affects t_i flags to invalidate the following bin decoding outputs.

Our proposed MBAD architecture is effectively designed to shorten the system critical path and to reduce area and power costs. In traditional CABAC engines, context derivation, AD decoding, renormalization and probability updating are processed in serial. If we directly map these data dependencies for multiple-bin decoding, it will result in a concatenated one-dimensional path with long computation delay. If we combine Fig. 7 or Fig. 9 with Fig. 10, we can find that in our proposed architecture, pStatus/context and range/offset propagation in horizontal direction and bin decoding in vertical direction can process in parallel. Our MBAD scheme efficiently maps the iterative one-dimensional arithmetic decoding algorithm into parallel two-dimensional architecture, and thus gains extremely high performance with relatively low area and power costs.

In addition, MBAD architecture is separated into several bin stages, and interconnections between stages are simple. So within the decoder core, most circuits are connected with local interconnections and the fan-out is relatively small too. This

feature can largely save redundant routing area and reduce the wire delay as well.

Renormalization is replaced by shifting one bit per cycle, so that the leading-one detection circuits and large shifter can be saved. Furthermore, the bit loading process can simultaneously conduct up to 16 bins, and the delay is relatively small compared with the core AD iterations.

V. Implementation Result

We have implemented the proposed architecture for TSMC 0.18μm process. The system critical path is the concatenated range update process during multiple MPS bin decoding. The maximum delay of critical path is about 22ns with total logic area of 42K gates (without context RAM). As we have analyzed in section III-D, system frequency requirement for real-time decoding is 42MHz. So we can decode the H.264 high definition level 4.0 in real time (see details in Table III).

TABLE III
Comparison of CABAC Decoders

Architectures	Process (μm)	Area (KGate)	Frequency (MHz)	Decoding Power
Chen's [11]	0.13	138*	200	CIF 25Hz
Yu's [12]	0.18	30	148.5	D1 30Hz
Kim's [19]	0.18	na	300	na
Eeckhaut's[20]	na	590ALMs	105	HDTV
Proposed	**0.18**	**42**	**45**	**1080i 30Hz**

* The area includes the context model SRAM.

To compare with our multi-bin scheme, we can directly expand the critical loops of [12] to decode multiple bins in one cycle. To guarantee decoding one bit per cycle for significance map and coefficient levels, the expansion factor should be 16 and the working frequency is as low as 9MHz which can not meet the real time requirement for HD applications.

Chen's design is based on 0.13μm technology. Experientially, to implement the circuits with the same gate count, 0.13μm implementation is about half in area compared with 0.18μm implementation; but the speed is almost the same because of the secondary effects of deep sub-micron circuits. Kim's architecture decodes one bin every three cycles, so the bin throughput is only 100M/s. Eeckhault implements the decoder on Altera's FPGA Stratix II S60, and the logic part occupies 590 Adaptive Logic Modules (ALM). He claims the decoder can support HD applications. But as shown in the analysis above, it can not guarantee the performance in the worst cases. Reconfigurable solution is flexible compared with ASIC, but the high cost and high power consumption features make it hard to become the mainstream in industry for video decoding applications. As for the average performance, our multiple-bin scheme can achieve as high as 102Mbin/s according to the simulation result on typical video streams. The average bin-rate is comparable with previous works, but we can guarantee the worst case performance because bit fluctuation is generally smaller than bin fluctuation in the real world.

Compared with previous works, our multiple-bin parallel decoding scheme does not induce large area cost. This is because that we have efficiently simplified the logic of every bin stage by taking the best advantages of constant-bit-rate strategy. MPS assumption reduces the bin selection computation from the exponential scale into the linear scale. And the circuits for generating new range are also simplified.

In addition, due to the low working frequency of our design, the power consumption is largely reduced compared with previous works. According to the equation $P = kCV^2 f$, we can estimate that the power consumption of our design is about 1/2.4 of Yu's and even less compared to other works. The power reduction is gained by our efficient variable-bin-rate scheme. Computation-intensive parts are paralleled using multiple-bin decoding scheme to reduce the working frequency. And low-loaded syntax elements are decoded using bin-by-bin scheme to avoid large increase in total area.

Our high performance CABAC decoding engine can be easily integrated into a video decoder chip. CABAC decoding is the first stage of video decoding. The following components, such as IDCT and motion compensation, generally adopt macroblock pipeline to achieve high performance. But the processing cycles used by CABAC decoding vary a lot for different macroblocks. We can place a frame-level buffer between CABAC and the following components to match the speed variation between CABAC and following components.

VI. Conclusion

This paper proposes a CABAC decoding engine for H.264 high definition real time decoding. Compared with the previous works, we give a thorough analysis on system performance requirement for real time decoding for the first time. This ensures our decoder engine can process real time decoding in the worst cases.

In our engine, we adopt the variable-bin-rate strategy. Constant-bit-rate scheme is used in MB data decoding to reduce the system frequency requirement. And constant-bin-rate scheme is used in header decoding to significantly reduce the system complexity for MBAD. The two schemes are combined to make our variable-bin-rate strategy very efficient.

Multiple-bin arithmetic decoding scheme benefits a lot from constant-bit-rate strategy. The MPS assumption is derived by constant-bit-rate strategy. And the assumption reduces the complexity of bin selection traversing computation from an exponential scale into a linear one. Thus the proposed MBAD scheme can efficiently map the iterative one-dimensional arithmetic decoding algorithm into parallel two-dimensional architecture, and gain extremely high performance with relatively low area and power costs.

Probability propagation modules are designed to facilitate the decoding of the four specific types of SEs. This minimizes the design complexity for each single PP module. Special characteristics of these SEs are efficiently utilized to optimize both context switching process and probability update process.

The implementation result shows that our decoder engine

can run up to 45MHz. This can guarantee that our decoder can process 1080i video of H.264 level 4.0 at 30 frames per second, even in the worst cases. Comparison result shows that the area increase of the proposed scheme is relatively small, whereas decoding power consumption is significantly reduced.

ACKNOWLEDGMENT

The authors would like to thank Di Wu, Junhao Zheng, Bing Sheng, Lei Deng, Huizhu Jia and etc. for useful discussions and valuable comments.

REFERENCES

[1] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, "Draft ITU-T recommendation and final draft international standard of joint video specification ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC," JVT_G050, 2003.

[2] T. Wiegand, G. J. Sullivan, G. Bjntegaard, A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol 13, no 7, pp. 560-576, Jul. 2003.

[3] J. Osterman, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, T. Wedi, "Video Coding with H.264/AVC: Tools, Performance and Complexity," *IEEE Circuits and Systems Magazine*, vol. 4, no. 1, pp. 7-28, Mar. 2004.

[4] D. Marpe, H. Schwarz, T. Wiegand, "Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC video compression standard", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no 7, pp. 620-636, Jul. 2003.

[5] K. Andra, C. Chakrabarti, T. Acharya, "A High-Performance JPEG2000 Architecture", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no 3, pp. 209-218, Jul. 2003.

[6] G. Pastuszak, "A Novel Architecture of Arithmetic Coder in JPEG2000 Based on Parallel Symbol Encoding", *IEEE Proc. Parallel Computing in Electronical Engineering*, pp. 303-308, Sept. 2004.

[7] J. L. Nunez, V. A. Chouliaras, "Design and Implementation of a High-Performance and Silicon Efficient Arithmetic Coding Accelerator for the H.264 Advanced Video Codec", *IEEE Proc. Application-Specific Systems, Architecture Processors*, pp. 411-416, Jul. 2005.

[8] J. L. Nunez, V. A. Chouliaras, "High-Performance Arithmetic Coding VLSI Macro for the H264Video Compression Standard", *IEEE Trans. Consumer Electronic*, vol. 51, no. 1, pp. 144-151, Feb. 2005.

[9] R. R. Osorio, J. D. Bruguera, "Arithmetic Coding Architecture for H.264/AVC CABAC Compression System", *IEEE Proc. Euromicro Symposium on Digital System Design*, pp. 62-69, Sept. 2004.

[10] R. R. Osorio, J. D. Bruguera, "A New Architecture for fast Arithmetic Coding in H.264 Advanced Video Coder", *IEEE Proc. Euromicro Conference on Digital System Design*, pp. 298-305, Sept. 2005.

[11] Jian-Wen Chen, Cheng-Ru Chang, Youn-Long Lin, "A Hardware Accelerator for Context-Based Adaptive Binary Arithmetic Decoding in H.264-AVC", *IEEE Proc. International Symposium on Circuits and Systems*, vol. 5, pp. 4525-4528, May 2005.

[12] Wei Yu, Yun He, "A high performance cabac decoding architecture", *IEEE Trans. Consum. Electron.*, vol. 51, no. 4, pp. 1352-1359, Nov. 2005.

[13] L. Bottou, P. G. Howard, Y. Bengio, "The Z-coder adaptive binary coder", *IEEE Proc. Data Compression Conference*, pp. 13-22, Apr. 1998.

[14] X. Marichal, B. Macq, M. P. Queluz, "Generic coder for binary sources: the M-coder", *IEE Electron. Lett.*, vol. 31, no. 7, pp. 544-545, Mar. 1995.

[15] J. L. Mitchell, W. B. Pennebaker, "Optimal Hardware and Software Arithmetic Coding Procedures for the Q-Coder", *IBM J. RES. Develop*, vol. 32, no. 6, pp. 727-736, Nov. 1988.

[16] M. J. Slattery, J. L. Mitchell, "Qx-coder", *IBM J. RES. Develop*, vol. 42, no. 6, Nov. 1988.

[17] S. F. Chang, D. G. Messerschmitt, "Designing High-Throughput VLC Decoder. I. Concurrent VLSI Architectures", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, no 2, pp. 187-196, Jun. 1992.

[18] H. D. Lin, D. G. Messerschmitt, "Designing a High-Throughput VLC Decoder. I. Parallel Decoding Methods", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, no 2, pp. 197-206, Jun. 1992.

[19] C. H. Kim, I. C. Park, "High Speed Decoding of Context-Bbased Adaptive Binary Arithmetic Codes Using Most Probable Symbol Prediction", *IEEE International Symposium on Circuits and Systems*, May 2006.

[20] H. Eeckhaut, M. Christiaens, D. Stroobandt, V. Nollet, "Optimizing of the Critical Loop in the H.264/AVC CABAC Decoder", *IEEE International Conference on Field Programming Technology*, pp. 113-118, Dec 2006.

Peng Zhang received the B.S. degree in electronic engineering and information science from University of Science and Technology of China, in 2002, and the M.S. degree in computer science from Institute of Computing Technology, Chinese Academy of Sciences, in 2004. At present, he is a Ph.D. candidate in Institute of Computing Technology, Chinese Academy of Sciences. His research interests include video coding, computer architecture, effective VLSI implementation and SoC design.

Don Xie received his Ph.D. degree in electrical engineering from University of Rochester, USA. He was a Senior Scientist at Eastman Kodak Company, New York, USA, from 1994 to 1997; a Principal Scientist at Broadcom Corporation, California, USA, from 1997 to 2002. He right now holds a position at Spreadtrum Communications, Inc. as a Senior Director. His research interests include: multimedia SoC design, embedded system. He holds 24 U.S. Patents.

Wen Gao (M'99) received the M.S. degree and the Ph.D. degree in computer science from Harbin Institute of Technology, Harbin, China, in 1985 and 1988, respectively, and the Ph.D. degree in electronics engineering from the University of Tokyo, Tokyo, Japan, in 1991. He was a Research Fellow with the Institute of Medical Electronics Engineering, University of Tokyo, in 1992, and a Visiting Professor with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, in 1993. From 1994 to 1995, he was a Visiting Professor with the Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge. Currently, he is the Director of the Joint R&D Lab (JDL) for Advanced Computing and Communication, Chinese Academy of Sciences, a Professor with the Institute of Computing Technology, a Professor of computer science with the Harbin Institute of Technology, and an honor Professor of computer science at City University of Hong Kong. He has published seven books and over 200 scientific papers. His research interests are in the areas of signal processing, image and video communication, computer vision, and artificial intelligence. Dr. Gao chairs the Audio Video coding Standard (AVS) workgroup of China. He is the head of the Chinese National Delegation to MPEG working group (ISO/SC29/WG11). He is also the Editor-in-Chief of the Chinese Journal of Computers and the general Co-Chair of the IEEE International Conference on Multi-model Interface in 2002.