A Highly Efficient VLSI Architecture for H.264/AVC Level 5.1 CABAC Decoder

Yuan-Hsin Liao, Gwo-Long Li, *Student Member, IEEE,* and Tian-Sheuan Chang, *Senior Member, IEEE*

***Abstract*—In this paper, a high throughput context-based adaptive binary arithmetic coding decoder design is proposed. This decoder employs a syntax element prediction method to solve pipeline hazard problems. It also uses a new hybrid memory two-symbol parallel decoding in order to enhance performance as well as to reduce costs. The critical path delay of the two-symbol binary arithmetic decoding engine is improved by 28% with an efficient mathematical transform. Experimental results show that the throughput of our proposed design can reach 485.76 Mbins/s in the high bit-rate coding and 446.2 Mbins/s on average at 264 MHz operating frequency, which is sufficient to support H.264/AVC level 5.1 real-time decoding.**

***Index Terms*—Binary arithmetic coding, CABAC, entropy decoding, H.264/AVC, syntax parsing.**

I. Introduction

H.264/AVC [1] IS THE state-of-the-art video coding standard developed to satisfy the demands for higher compression efficiency induced by the growing popularity of high definition (HD) TV. With a number of advanced techniques, H.264/AVC can provide better compression efficiency compared to the earlier MPEG-4 and H.263 standards, and has been widely used in the video application systems. The enhanced entropy coding algorithm improves the coding efficiency and makes a major contribution. However, its intensive and serial computations impose a significant overhead on entropy decoder and consequently make decoding difficult to work in real-time. As a result, a hardware accelerator for entropy decoding is indispensable for a real-time HD H.264/AVC video decoder.

In H.264/AVC, two entropy coding methods are specified: context-based adaptive variable length coding (CAVLC), and context-based adaptive binary arithmetic coding (CABAC) [1], [2]. Both methods employ context-based adaptive modeling in their entropy coding framework to achieve better compression efficiency compared to previous video coding standards. Since CABAC provides an adaptation to non-stationary symbol statistics and has no limitation of integer bits/symbol, it can achieve 9% to 14% bit-rate reduction on average in comparison to CAVLC [3]. However, the inherently strong data dependency significantly restricts the throughput of CABAC decoder and it is generally considered as the main design challenge in the hardware implementation.

In this paper, a CABAC decoder design with the properties of high throughput and low hardware cost is proposed. First, the characteristics of syntax element parsing flow and bin distribution among syntax elements are analyzed. Based on the observed statistics, we propose a prediction-based two-stage pipelining decoding architecture which combines syntax element parsing and decoding with a new hybrid memory design to circumvent the syntax element switching overhead problem at a reasonable hardware cost. In addition, further improvement in throughput is achieved by a two-symbol parallel decoding technique. Through our proposed architecture, the decoding performance is significantly boosted, especially in the high bit-rate coding.

The remainder of this paper is organized as follows. In Section II, we briefly describe the CABAC decoding and its design challenges in the hardware implementation. The prediction scheme and the hardware architecture design are presented in Sections III and IV, respectively. Finally, we provide simulation results in Section V to demonstrate the performance of our design and draw a conclusion in Section VI.

II. CABAC Decoding and Previous Works

In CABAC, every syntax element (SE) is composed of a series of bins. These bins will be decoded into their original values by CABAC decoder when combined with syntax element parsing. In the parsing flow, each SE is parsed sequentially. After the type of SE is decided, the corresponding bin decoding process is executed depending on the bin index. Finally, the constructed bin string is de-binarized. If any codeword is matched, the corresponding value is returned and the decoding procedure for current SE is complete.

In the hardware realization, the bin decoding process consists of four elementary steps: context selection (CS), context model loading (CL), binary arithmetic decoding (BAD), and binarization matching (BM). To read the specific context model from the context model memory, the memory addresses must be calculated first. Generally, the memory address of each context model is the same as its corresponding context index.

Manuscript received September 14, 2010; revised April 26, 2011; accepted June 11, 2011. Date of publication June 27, 2011; date of current version February 8, 2012. This work was supported by the National Science Council of Taiwan, under Grant NSC98-2220-E-009-001. This paper was recommended by Associate Editor J. Ridge.

Y.-H. Liao is with PixArt Imagine, Inc., Hsinchu 300, Taiwan (e-mail: ashin29@gmail.com).

G.-L. Li and T.-S. Chang are with the Department of Electronics Engineering, National Chiao-Tung University, Hsinchu 300, Taiwan (e-mail: glli@dragons.ee.nctu.edu.tw; tschang@twins.ee.nctu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2011.2160752

TABLE I
IMPROVEMENT OF PREDICTION ACCURACY USING THE PROPOSED METHODS (FOR IPPP FRAME STRUCTURE)

Video Sequence (IPPP)	QP	SE Switching Rate (%)	Hit Rate (%)	
			Without SE Merging	With SE Merging
Pedesttrian_area	28	65.95	79.47	92.65
	20	75.38	79.48	96.60
	12	72.24	79.09	99.39
Riverbed	28	69.36	80.85	97.53
	20	68.73	81.98	98.72
	12	62.66	82.54	99.70
Rush_hour	28	55.23	79.80	91.63
	20	67.77	79.39	95.57
	12	73.98	78.60	99.29
Station2	28	63.73	79.67	91.67
	20	79.36	78.20	96.77
	12	73.15	79.17	99.53
Sunflower	28	61.66	79.77	90.46
	20	67.01	77.80	93.09
	12	74.95	78.02	98.76
Tractor	28	64.46	77.50	93.13
	20	75.01	78.08	97.78
	12	67.86	81.16	99.71
Average		68.81	79.48	96.22

TABLE II
IMPROVEMENT OF PREDICTION ACCURACY USING THE PROPOSED METHODS (FOR IBBBP FRAME STRUCTURE)

Video Sequence (IBBBP)	QP	SE Switching Rate (%)	Hit Rate (%)	
			Without SE Merging	With SE Merging
Pedesttrian_area	28	64.11	79.65	92.14
	20	72.68	79.4	96.11
	12	72.3	78.93	99.38
Riverbed	28	67.09	80.66	97.24
	20	67.87	81.97	98.69
	12	62.47	82.54	99.7
Rush_hour	28	54.4	80.68	90.98
	20	64.16	79.35	94.71
	12	73.73	78.37	99.23
Station2	28	61.45	80.58	91.88
	20	75.94	78.23	96.03
	12	73.82	78.84	99.53
Sunflower	28	61.46	79.96	89.99
	20	64.56	78.7	93.08
	12	73.89	78.59	98.61
Tractor	28	63.18	78.02	93.08
	20	72.16	79.07	97.71
	12	66.78	81.31	99.72
Average		67.34	79.71	95.99

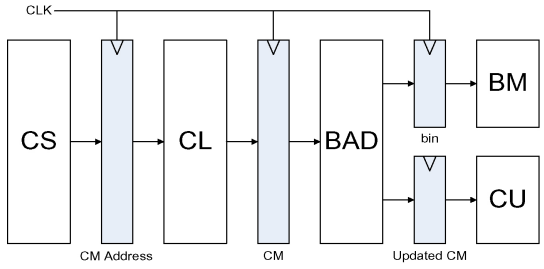

Fig. 1. Pipeline scheme of CABAC decoding.

However, the organization of context models in H.264/AVC is clearly not the most economical, and thus needs to be reorganized for better performance. After the context model (CM) is loaded from CM memory according to the address, a bin is decoded to be the most probable symbol or least probable symbol (MPS or LPS) based on the probability model provided by the CM in the BAD stage. Afterward, the constructed bin string is de-binarized in the final stage to decide whether the decoding process of current SE is finished or not, and the context model update (CU) process takes place at the same time.

Since every bin is decoded by the same chain of operations (CS→CL→BAD→BM and CU), the decoding performance can be improved by exploiting the pipelining scheme as presented in [4]. Fig. 1 shows a 4-stage bin-level pipelining CABAC decoder design. However, throughput improvements are limited by the pipeline hazard coming from SE switching. In some works such as [4]–[8], the proposed architectures only focused on the bin-level decoding process, while leaving SE parsing to another processor. Although the separation of SE parsing and decoding makes the implementation of CABAC decoding much simpler, it results in that the actual throughput cannot reach its theoretical maximum, since the context model has to be reloaded and the pipeline is thus stalled whenever SE switching takes place. For example, as discussed in [9], if only intra-SE (CABAC decoding is limited in a single SE) decoding is taken into account, the pipeline architecture of [4] can work without any stall and thus achieve almost 1 cycle/bin decoding performance. However, when it works in practice where the inter-SE (SE switching in the parsing flow) switching must be handled, the decoding performance degrades to about 4 cycles/bin due to frequent pipeline stall. This significant performance drop indicates the fact that performing great performance in intra-SE decoding does not mean it will do as well in inter-SE decoding. Tables I and II show the average SE switching rate for different frame structures. In which SE switching rate can be calculated by the number of decoded SEs/the number of decoded bins. From this table, we observed that the SE switching rate is up to 67.34% on average. Therefore, such noticeable SE switching rate will significantly degrade the decoding performance. To relieve the performance degradation originated in SE switching overhead, a SE predictor was proposed in [9], where the correlation between successive SEs was exploited in order to achieve higher prediction accuracy when compared to MPS prediction. Chang [10] described a clustering method to lower the frequency of CM switching.

Besides pipelining, multisymbol decoding architecture is also generally considered as a helpful method to speed up the decoding procedure. Yu and He [11] proposed a multisymbol decoding architecture which can decode 1 to 3 bins per cycle. A two-symbol parallel decoding method was proposed to enhance the decoding performance by predicting whether the

TABLE III
STATISTICAL RESULTS OF BIN DISTRIBUTION (FOR IPPP FRAME STRUCTURE)

Syntax Element	Video Sequence (IPPP_QP20)						Average (%)
	Pedesttrian_area	*Riverbed*	*Rush_hour*	*Station2*	*Sunflower*	*Tractor*	
mb_type	1.49	0.54	2.51	1.51	3.65	0.95	1.78
mb_skip_flag	0.59	0.25	0.92	0.57	1.64	0.34	0.72
intra_pred_mode	2.75	4.17	3.16	0.99	1.44	0.7	2.2
mvd	3.8	0.1	11.69	5.46	11.98	4.79	6.3
coded_block_pattern	3.04	1.56	4.46	2.87	6.93	1.95	3.47
coded_block_flag	4.79	2.92	3.92	4.86	5.39	4.03	4.32
significant_coeff_flag	39.71	33.49	32.7	45.85	26.74	39.4	36.32
last_significant_coeff_flag	12.06	13.99	10.88	11.24	11.08	13.8	12.18
coeff_abs_level_minus1	29.48	41.75	26.58	24.53	27.17	32.71	30.37

first symbol is MPS [12]. Xu *et al*. [13] applied a tree decoding structure to *coeff_abs_level_minus1*, which can decode at most three bins within the same cycle. The architecture in [14] employed a branch selection two-symbol parallel decoding technique to resolve data dependency problem. This architecture can process two bins within one cycle for general cases but suffers from high area cost. Zhang *et al*. [15] presented a variable-bin-rate CABAC engine that can produce up to 16 bins at the same time. A look-ahead decision parsing technique was proposed to speed up CABAC decoding in [16]. Chen and Lin [17] proposed a fully hardwired CABAC decoder which is able to decode at most two bins in one cycle for certain syntax elements: *coeff_abs_level_minus1*, *significant_coeff_flag*, *last_significant_coeff_flag*, and *mvd*.

III. PROPOSED PREDICTION SCHEME

A. SE Parsing Flow and Bin Distribution Analyses

In this section, the factors that dominate the overall entropy decoding performance are discussed first. If common factors can be found, we can adjust our design to those cases for achieving better decoding performance. The characteristics of SE parsing flow and bin distribution among SEs are analyzed as follows.

The SE parsing flow depends mainly on conditional branches as illustrated in Fig. 2. Branches denoted by "*" indicate that the condition of branch and the current SE value are independent. In other words, the next SE type to be decoded right after current SE can be decided before the current SE decoding is completed. Therefore, for this kind of branches, the context models used for decoding the next SE can be prepared in advance to prevent pipeline stalling. However, most branches denoted by "#" are dependent on the current SE value. In these branches, the next SE type cannot be determined until the current SE value is ascertained. Take the significance map (*significant_coeff_flag* and *last_significant_coeff_flag*) as example, the SE right after *significant_coeff_flag* may be *significant_coeff_flag* or *last_significant_coeff_flag* depending on the decoded value as shown in Fig. 3.

Tables III and IV list the analyzed results of bin distribution based on the video sequences with HD 1920×1080, frame rate 30 f/s, coding structure IPPP/IBBBP, and QP20 by

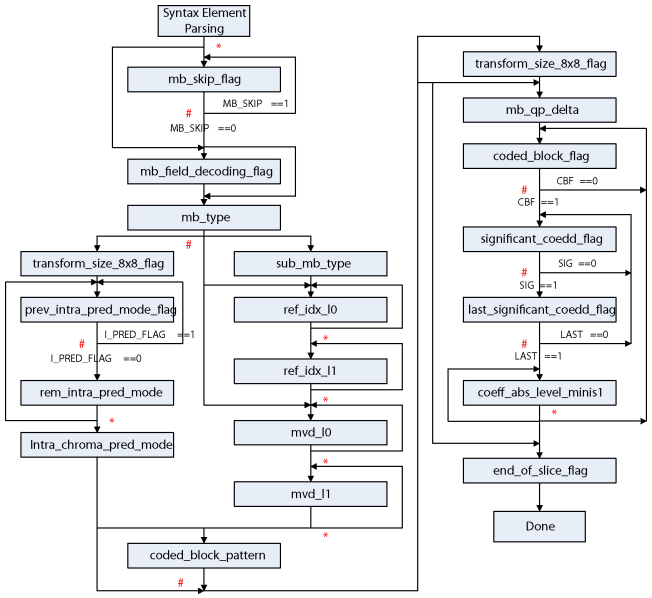

Fig. 2. SE parsing flow for the H.264/AVC.

using JM reference software 12.2 [18]. From the statistics, we can observe that the proportion of *significant_coeff_flag* and *last_significant_coeff_flag* reaches up to 45% at least of total bins and the portion of *coeff_abs_level_minus1* reaches up to 30% on average. However, only *significant_coeff_flag* and *last_significant_coeff_flag* have been involved in our CABAC decoder design to solve the data hazard problem, since the 1-bin SEs of *significant_coeff_flag* and *last_significant_coeff_flag* have 100% SE switching rate between them, and each one is decoded at most once in the same significance map. Nevertheless, as for *coeff_abs_level_minus1*, although the portion of *coeff_abs_level_minus1* reaches up to 30% on average, they usually contain several bins especially in high bit-rate coding, which result in low SE switching rate and thus helpless for solving the problem of data hazard.

B. SE Prediction

To reduce the performance degradation while avoiding being burdened with hardware cost overhead, we propose an efficient approach to predict the type of the SE in current MB. The value of the SE in current MB is predictable by referring to the

TABLE IV
STATISTICAL RESULTS OF BIN DISTRIBUTION (FOR IBBBP FRAME STRUCTURE)

Syntax Element	Video Sequence (IBBBP_QP20)						Average (%)
	Pedesttrian_area	*Riverbed*	*Rush_hour*	*Station2*	*Sunflower*	*Tractor*	
mb_type	2.37	1.45	4.1	2.48	3.95	1.39	2.62
mb_skip_flag	0.67	0.25	1.04	0.74	1.71	0.35	0.79
intra_pred_mode	3.19	4.02	2.98	1.6	2.12	1.32	2.54
mvd	3.39	0.33	13.41	6.74	11.05	4.47	6.57
coded_block_pattern	3.24	1.55	4.77	3.2	6.44	1.99	3.53
coded_block_flag	5.33	2.9	4.48	5.59	5.49	4.3	4.68
significant_coeff_flag	35.68	33.01	28.04	40.48	23.73	35.81	32.79
last_significant_coeff_flag	12.16	13.85	10.62	11.12	11.25	13.87	12.15
coeff_abs_level_minus1	31.07	41.4	26.44	24.97	29.29	34.86	31.34

SEs in neighboring MB, since the high correlation between the features of image in spatial domain. Therefore, by assuming that the value of the SE is very similar to its neighbor, we use the SEs of left MB to predict next SE of current MB. For example, in the flow shown in Fig. 2, when decoding *mb_type* SE of current MB, the *mb_type* SE of left MB is used to predict whether the next SE of current MB would be *transform_size8×8_flag*, *sub_mb_type*, or *mvd*. Through this approach, the performance can be improved. The prediction accuracy of this method can achieve 79.48% on average as shown in Tables I and II. Note that Hit Rate = (the number of prediction hits)/(the number of decoded SEs). In practice, since the corresponding SE registers would not be updated before the current SE has been successfully decoded, the left MB SE values are temporally stored in the SE registers in the raster scan decoding order. As a result, no additional memory spaces are required to store the SEs of left MB in our design.

To further improve the accuracy of prediction, we merge all symbols of the significance map which is composed of *significant_coeff_flag* and *last_significant_coeff_flag* as an individual SE by exploiting their decoding regularities since *significant_coeff_flag* and *last_significant_coeff_flag* account for the majority of prediction misses. More precisely, after applying the SE merging, the significance map contains all *significant_coeff_flag* and *last_significant_coeff_flag* of a residual block (4×4 or 8×8), in which *bin[2×i]* and *bin[2×i+1]* represent *significant_coeff_flag[i]* and *last_significant_coeff_flag[i]*, respectively. The benefit of SE merging is that the inter-SE (SE switching in the parsing flow) issue is transformed into an intra-SE issue (CABAC decoding is limited in a single SE). As a result, predictions for the branches right after *significant_coeff_flag* and *last_significant_coeff_flag* are not necessary anymore. What we have to care about is the rule of bin index transition, which can be summarized in Table V. Note that the binarization matching of significance map will be satisfied on condition that the current bin is *last_significant_coeff_flag* and its bin value is 1, or the current bin index meets the final bin index *(((binIdx % 2 = = 1) && (binVal = = 1)) ||(binIdx = = numCoeff - 1))*.

Compared with the predictor which does not realize SE merging, the combination of SE merging and SE prediction method can achieve about 17% higher prediction accuracy on average, as shown in Tables I and II. Moreover, in the high bit-rate coding such as QP equaling to 12, the prediction accuracy can reach over 99% for almost all test sequences.

8	0	1	1
-3	0	0	0
0	0	0	0
0	0	0	0

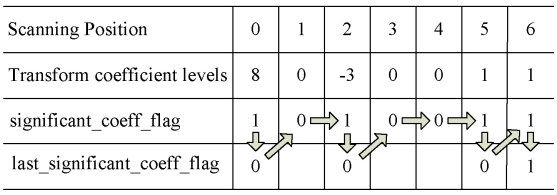

Fig. 3. Example for decoding the significance map of a 4×4 residual block.

TABLE V
BIN INDEX TRANSITION RELATION IN SIGNIFICANCE MAP

Current Flag	Bin Value	Next Flag	Next *binIdx*
SIG[*i*]	0	SIG[*i*+1]	*binIdx* + 2
SIG[*i*]	1	LAST[*i*]	*binIdx* + 1
LAST[*i*]	0	SIG[*i*+1]	*binIdx* + 1
LAST[*i*]	1	X	X

i denotes scanning position.
SIG denotes significant_coeff_flag.
LAST denotes last_significant_coeff_flag.

IV. DESIGN OF CABAC DECODING ARCHITECTURE

Fig. 4 shows the system level architecture of proposed CABAC decoder. At the start of each slice, CABAC decoder and all CMs are initialized first. Afterward, at each decoding point of the parsing flow as shown in Fig. 2, the SE parser transmits a corresponding SE type and a predicted next SE type with a request signal to invoke the CABAC decoder. Once the binarization matching condition is satisfied, the CABAC decoder outputs a decoded SE value to the SE Parser. Some SEs which involve in the parsing flow decision in the future are kept in the SE register; others are sent to the neighbor manager for further usage such as CS and inverse transform.

Since it is obvious that the main obstacle to adopt pipelining scheme for CABAC decoder comes from data hazards, the

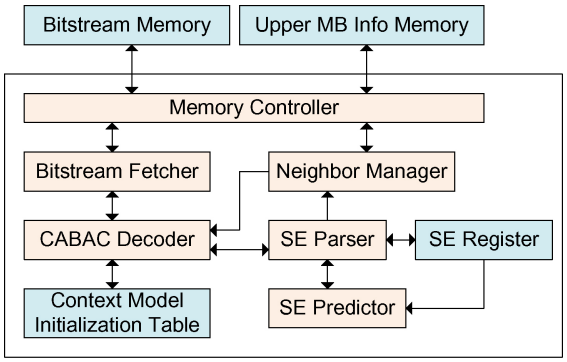

Fig. 4. Proposed CABAC decoder architecture.

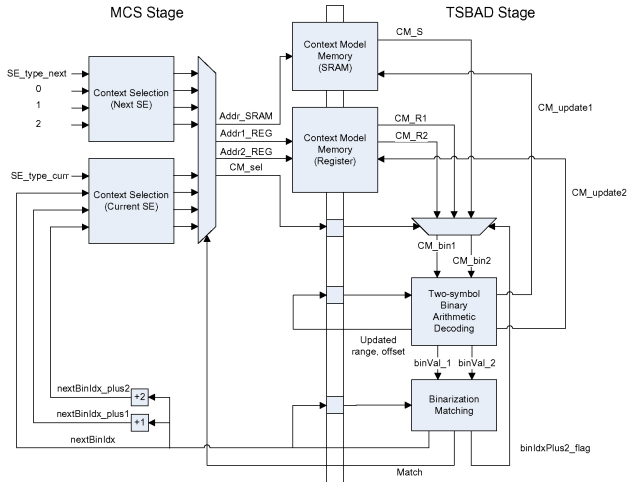

Fig. 5. Essential parts of our proposed CABAC decoder engine.

design of pipelining stages shall be considered carefully. With the hardware-oriented SE prediction technique, we design a prediction-based two-stage pipelining architecture as shown in Fig. 5. In our architecture, the chain of operations is divided into two stages, modified context model selection (MCS) stage and two-symbol binary arithmetic decoding (TSBAD) stage. The MCS stage contains CS and CL. Fig. 6 exhibits the block diagram of the proposed TSBAD in which two-symbol binary architecture decoding engine and CU are included. In H.264/AVC video coding standard, the bin is a basic decoding unit of CABAC decoder, and there are three types of decoding processes to derive a bin: non-equiprobable decoding (to derive regular bin), bypass decoding (to derive bypass bin), and terminate decoding (to derive terminate bin). Our proposed TSBAD architecture is able to decode two regular bins, one regular bin and one bypass bin, one regular bin and one terminate bin, or two bypass bins at most in one cycle. The details will be discussed in the following subsections.

A. Modified Context Model Selection

The main idea of MCS stage is to select CMs for decoding the next two bins. To match the property of SE parser that can only parse SEs one by one, we restrict the two-symbol decoding to a single SE only. After the SE merging is employed in the decoding framework, for successive two bins

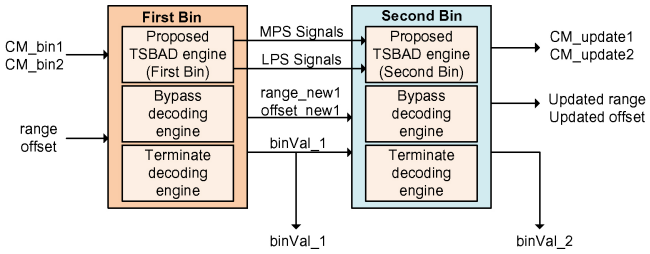

Fig. 6. Block diagram of proposed TSBAD.

to be decoded, the position of the second bin in a SE may be *binIdx_plus1* (*binIdx* + 1) or *binIdx_plus2* (*binIdx* + 2). It means that by giving two possible CMs, the second bin can be decoded according to the necessary CM chosen by its actual bin index. Furthermore, by means of the prediction mechanism, the CMs of predicted next SE and the CMs of current SE can be calculated in parallel. In the end the value of current SE is confirmed, the CABAC decoder can keep processing without stall if the actual result matches what we presume. Otherwise, the pipeline has to be stalled for recalculating the context models of next SE. In our proposed two-stage pipelining architecture, the penalty of prediction miss is merely one cycle as illustrated in Fig. 7.

With the prediction scheme, we employ two CS modules to calculate SRAM memory address (*Addr_SRAM*) and register memory addresses (*Addr1_REG* and *Addr2_REG*) in parallel, one for the current SE and another one for the predicting next SE. For the CS module of next SE, only bin indexes 0, 1, and 2 are accounted, since a predicted next SE will be transformed into a current SE if the prediction hits in the next cycle. Then, the CS module of current SE continues decoding the predicted next SE for bin indexes larger than 2 and the CS module of next SE starts processing with new bins. In other words, the CS module of next SE only needs bin indexes of 0, 1, and 2 as the inputs all the time. The CS module of current SE will be in charge of the calculation for bin indexes which are larger than 2. As a result, instead of duplicating the hardware to satisfy the requirement that calculating CM memory addresses for the current SE and the next SE at the same time, unnecessary calculation in the prediction module is removed and the hardware cost overhead is thus suppressed. Finally, the result of BM will determine which one is chosen for CL. Furthermore, because the CM provided for the first bin decoding (*CM_bin1*) may come from the SRAM or the Register port 1 (*CM_S* or *CM_R1*), and CM provided for the second bin decoding (*CM_bin2*) may come from the Register port 1 or the Register port 2 (*CM_R1* or *CM_R2*), an additional selective signal (*CM_sel*) which is SE-dependent is also transmitted from the MCS stage to the TSBAD stage.

B. Context Model Memory Design

In the proposed MCS stage design, to reach the target of loading three specific CMs and storing updated CMs in the same cycle, the design of CM memory shall be considered carefully. On the premise that one clock domain is used, the design of CM memory becomes a tradeoff between decoding performance and hardware cost. The first way to implement

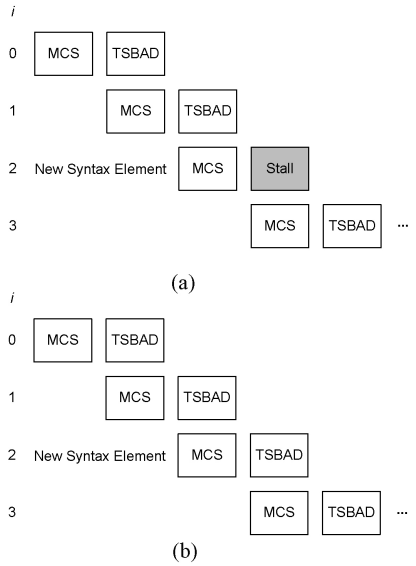

Fig. 7. Pipeline scheduling of (a) prediction miss and (b) prediction hit.

CM memory is to use single-port SRAM. The advantage of single-port SRAM is its low hardware cost. However, single-port SRAM cannot execute read operation and write operation simultaneously. Therefore, the operations of CL and CU have to be separated, which results in one extra cycle. Yi *et al.* [4] proposed a cache-liked context model reservoir (CMR) structure to resolve the conflict between CL and CU caused by structure hazard. Several context models that are probably used are cached in CMR. This allows the decoder to postpone CU and enables the parallel processing of CS and CL. Although the CMR structure is effective, the decoding procedure still has to be stalled by 2 cycles whenever CMR switching takes place.

Another way to implement CM memory is to use dual-port SRAM. The hardware cost of dual-port SRAM is two times larger than that of single-port SRAM in general. In spite of the advantage that the read operation and the write operation can be executed in the same cycle, frequent CMR switching while processing the significance map still significantly degrades the decoding performance. A context table reallocation scheme was presented in [17] to read two CMs at once by dividing the CM memory into two parts: a general context memory and extended context memory. However, it does not always work since the reallocation is only designed for specific SEs.

Storing all CMs in registers is the most convenient way to implement the CM memory due to the access of CMs would not limited by the number of ports like storing in single-port SRAM or dual-port SRAM. Nevertheless, the expense of hardware cost is too high. Thus, we propose a more suitable approach to implement the CM memory with hardware cost consideration while maintaining the decoding performance. In the proposed decoding framework, since the two-symbol decoding procedure is restricted to a single SE only, the CL for some SEs is simple such as flag-type SE, in which only one context model (*CM_bin1*) is necessary. For example, there are three candidate CMs used for decoding *transform_size_8x8_flag*. However, only one of them is necessary for TSBAD since *transform_size_*8 × 8*_flag* is composed of one single bin. Thus, the CL for the second

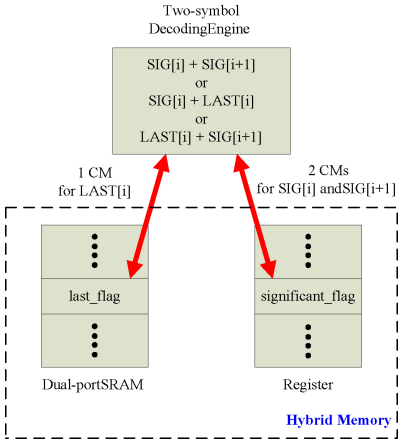

Fig. 8. Memory access of significance map decoding.

bin can be skipped, and only one CM for the first bin has to be concerned. For this type of SEs, a dual-port SRAM is sufficient to support CL and CU. However, for the other SEs like significance map, the CL is much more complicated. When decoding significance map, the next two bins to be decoded may be two successive *significant_coeff_flags* (*SIG*[*i*], *SIG*[*i* + 1]), one *significant_coeff_flag* followed by one *last_significant_coeff_flag* (*SIG*[*i*], *LAST*[*i*]), or one *last_significant_coeff_flag* followed by one *significant_coeff_flag* (*LAST*[*i*], *SIG*[*i* + 1]). Therefore, two CMs of *significant_coeff_flag* CM set and one CM of *last_significant_coeff_flag* CM set have to be loaded from CM memory concurrently. Moreover, two of them must be updated and write back. Therefore, all register based memory is the best solution due to the limitation of the number of ports of SRAM

Fortunately, for the different complexities of CL, it is reasonable to load CMs from different sources and assign them to the TSBAD stage according to the SE type and the bin indexes of next two bins. As a result, we reorganize the 459 CMs by applying the following principle. For every set of CMs, if two CMs are never used for TSBAD simultaneously, it is stored in a dual-port SRAM; otherwise, it is stored in registers. For instance, to satisfy the requirement for loading three CMs (one for *last_significant_coeff_flag* and two for *significant_coeff_flag*) from the CM memory and execute storing operation in the same cycle, *significant_coeff_flag* CM set is stored in register while *last_significant_coeff_flag* CM set is stored in SRAM as depicted in Fig. 8. Guided by the principle, the organization of CM memory is listed in Tables VI and VII. Our proposed hybrid CM memory, half of dual-port SRAM and half of register, not only avoids structure hazards caused by CM reading and writing but also reduces the hardware cost overhead significantly in comparison to the implementation of all register approaches.

C. Critical Path Improvement in the TSBAD Stage

In the TSBAD stage, *CM_bin1* and *CM_bin2* provided for the first bin decoding and the second bin decoding is chosen by the selective signal (*CM_sel*) first. Afterward, in the bin decoding procedure, the updated CMs (*CM_update1*

TABLE VI
CONTENT OF SRAM AFTER REORGANIZATION OF OUR PROPOSAL

Address	CM Index	Syntax Element
0–2	0–2	mb_type (SI)
3–5	11–13	mb_skip_flag (P/SP)
6–8	24–26	mb_skip_flag (B)
9–11	70-72	mb_field_decoding_flag
12–31	85–104	coded_block_flag
32–171	166–226, 338–398, 417–425, 451–459,	last_significant_coeff_flag
172–201	227–231, 237–241, 247–251, 257–261, 266–270, 426–430,	coeff_abs_level_minus1 (first bin)
202–204	399–401	transform_size_8 × 8_flag

TABLE VII
CONTENT OF REGISTER AFTER REORGANIZATION OF OUR PROPOSAL

Address	CM Index	Syntax Element
0–7	3–10	mb_type (I)
8–14	14–20	mb_type (P/SP)
15–17	21–23	sub_mb_type (P/SP)
18–26	27–35	mb_type (B)
27–30	36–39	sub_mb_type (B)
31–44	40–53	Mvd
45–50	54–59	ref_idx
51–54	60–63	mb_qp_delta
55–58	64–67	intra_chroma_pred_mode
59	68	prev_intra_pred_mode_flag
60	69	rem_intra_pred_mode
61–72	73–84	coded_block_pattern
73–224	105–165, 277–337, 402–416, 436–450,	significant_coeff_flag
225–253	232–236, 242–246, 252–256, 262–265, 271–275, 431–435,	coeff_abs_level_minus1 (first bin excluded)

and *CM_update2*) are written back into CM memory, and the decoding parameters, interval range and coding offset, are both refreshed. Eventually, the values of two bins are passed to the BM module to derive the value of SE and check whether the current SE decoding is done or not.

Following the two-symbol binary arithmetic decoding engine, the final step of this stage is the BM process that maps the constructed binary sequence to non-binary value. Therefore, the main critical path of this stage occurs in the bin value decision process of TSBAD engine. According to the H.264/AVC standard, the bin value decision is dependent on O_{LPS}. If O_{LPS} is negative, the *binVal* is identified as MPS; otherwise, the *binVal* is identified as LPS. For O_{LPS} to be calculated, a sequential procedure is defined in the standard like Fig. 9(a) shows. To obtain R_{MPS}, it is necessary to run through a 256-to-1 multiplexer first and then do the subtraction. However, a mathematical reordering method [7] can be adopted as follows:

$$O_{\mathrm{LPS}} = O - R_{\mathrm{MPS}} = O - (R - R_{\mathrm{LPS}}) = (O - R) + R_{\mathrm{LPS}}. \quad (1)$$

In (1), although $R_{\mathrm{MPS}} = R - R_{\mathrm{LPS}}$ cannot be obtained until R_{LPS} is selected by accessing the look-up table. However, since both R and O are ready in the beginning, the computation of $(O - R)$ and the table look-up for R_{LPS} can be operated in parallel. As a result, benefited by the calculation reordering, a balanced structure can be achieved for reducing the delay of bin value decision process as depicted in Fig. 9(b).

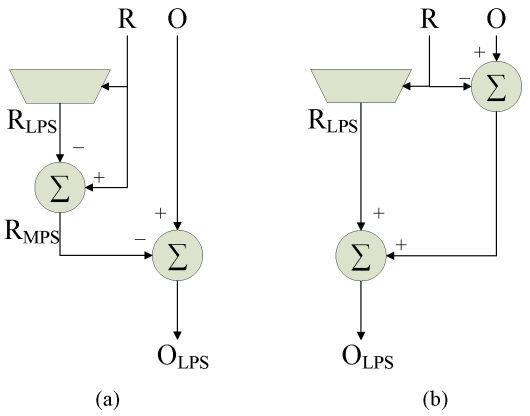

Fig. 9. Mathematical reordering from [7]. (a) O-(R-R_{LPS}). (b) (O-R)+R_{LPS}.

if previous bin is MPS **then**

$$\begin{aligned} O'_{LPS} &= (O_{MPS} - R_{MPS}) + R'_{LPS} \\ &= (O - R_{MPS}) + R'_{LPS} \\ &= O_{LPS} + R'_{LPS} \end{aligned}$$

else

$$\begin{aligned} O'_{LPS} &= (O_{LPS} - R_{LPS}) + R'_{LPS} \\ &= (O - R + R_{LPS} - R_{LPS}) + R'_{LPS} \\ &= (O - R) + R'_{LPS} \end{aligned}$$

Fig. 10. Mathematical transform for the second bin decision process.

Despite the fact that the mathematical reordering shortens the critical path delay of bin value decision process, multi-bin decoding based on cascaded binary arithmetic decoders still leads to an unavoidably long critical path, and thus remains unsuitable for the hardware implementation. The main barrier to exploit parallel decoding is inter-bin dependency of range (R) and offset (O). In our approach, instead of straight cascading binary arithmetic decoders, we extend the concept of (1) to two-symbol two-stage computation. According to (1), we execute the mathematical transform for the second bin decision process as shown in Fig. 10, where R'_{LPS} and O'_{LPS} represent R_{LPS} and O_{LPS} of the second stage, respectively. For the reason that O_{LPS} and $(O - R)$ are already calculated in the first bin decision process, the delay of a subtractor can be further eliminated.

Fig. 11 shows the detailed architecture of the proposed TSBAD engine. In the first bin decision scheme, state index (*stateIdx*) and MPS value (*valMPS*) are extracted from *CM_bin1*. The parameters with word "*renorm*" denote that they are left-shifted by the renormalization process. The shift amount of MPS case is 0 or 1 depending on the most significant bit of R_{MPS}, whereas the shift amount of LPS case lies in the range 1 to 7. To pass the shifted O_{LPS} and $(O - R)$ to the second bin decision scheme as soon as possible, a table-driven selector is adopted to derive the

TABLE VIII
CABAC DECODING PERFORMANCE OF THE PROPOSED ARCHITECTURE (FOR IPPP FRAME STRUCTURE)

Video Sequence (IPPP)	QP	Bitrate (Mb/s)	Throughput (bin/s)	Decoding Cycle	Penalty	Optimal Decoding Speed (bin/cycle)	Actual Decoding Speed (bin/cycle)
Pedestrian_area	28	6.24	8 243 604	5 096 931	399 656	1.617	1.5
	20	29.54	39 925 740	22 531 263	1 021 723	1.772	1.695
	12	125.65	167 323 465	90 263 065	734 579	1.854	1.839
Riverbed	28	25.6	33 793 858	19 033 689	577 882	1.775	1.723
	20	69.18	93 001 166	50 914 635	815 699	1.827	1.798
	12	170.25	229 139 086	123 698 211	431 383	1.852	1.846
Rush_hour	28	4.23	5 759 310	3 571 634	266 172	1.613	1.501
	20	19.46	25 637 361	14 569 250	769 343	1.76	1.671
	12	108.29	148 073 162	79 495 164	777 936	1.863	1.845
Station2	28	2.97	4 323 558	2 776 874	229 353	1.557	1.438
	20	29.02	41 714 033	23 398 501	1 070 737	1.783	1.705
	12	131.7	176 574 564	95 089 541	602 285	1.857	1.845
Sunflower	28	2.9	3 913 585	2 522 353	230 268	1.552	1.422
	20	11.51	14 473 353	8 612 806	670 697	1.68	1.559
	12	87.76	116 421 680	63 687 024	1 082 948	1.828	1.797
Tractor	28	10.6	13 428 740	7 986 431	594 672	1.681	1.565
	20	52.77	70 588 463	38 960 452	1 173 367	1.812	1.759
	12	155.65	203 467 876	109 714 968	405 875	1.855	1.848
Average						1.75	1.69

Optimal decoding speed = throughput/decoding cycle.
Optimal decoding speed = throughput/(decoding cycle + penalty).

TABLE IX
CABAC DECODING PERFORMANCE OF THE PROPOSED ARCHITECTURE (FOR IBBBP FRAME STRUCTURE)

Video Sequence (IBBBP)	QP	Bitrate (Mb/s)	Throughput (bin/s)	Decoding Cycle	Penalty	Optimal Decoding Speed (bin/cycle)	Actual Decoding Speed (bin/cycle)
Pedestrian_area	28	6.18	7 996 797	4 963 851	403 129	1.611	1.49
	20	26.48	33 865 563	19 296 701	957 390	1.755	1.672
	12	122.27	158 460 245	85 712 717	706 222	1.849	1.834
Riverbed	28	25.95	33 157 233	18 739 109	613 915	1.769	1.713
	20	69.51	90 186 496	49 414 413	799 758	1.825	1.796
	12	170.56	221 804 661	119 855 332	419 247	1.851	1.844
Rush_hour	28	3.96	5 359 183	3 361 522	262 862	1.594	1.479
	20	17.3	21 880 212	12 574 561	742 208	1.74	1.643
	12	106.46	141 397 927	76 048 399	799 083	1.859	1.84
Station2	28	3.09	4 487 035	2 896 824	223 772	1.549	1.438
	20	22.71	30 697 181	17 486 212	924 615	1.756	1.667
	12	124.92	164 313 863	88 365 185	565 193	1.859	1.848
Sunflower	28	2.81	3 747 747	2 425 104	230 656	1.545	1.411
	20	10.95	13 361 034	7 942 264	596 995	1.682	1.565
	12	81.97	106 090 796	58 346 381	1 085 765	1.818	1.785
Tractor	28	10.2	12 561 227	7 479 415	548 993	1.679	1.565
	20	51.34	65 822 524	36 557 868	1 087 536	1.801	1.748
	12	152.29	194 679 968	104 946 453	362 293	1.855	1.849
Average						1.74	1.68

shift amount of LPS case. In the second bin decision scheme, both cases for previous bin being MPS and LPS are calculated in parallel. With regard to *CM_bin2*, it has to be set to the updated *CM_bin1* when *CM_bin2* and *CM_bin1* are the same. For the reason that the second bin decision process is a parallel working, on the premise that knowing what previous bin is, instead of waiting the updated value of *CM_bin1* is determined, we can access the R_{LPS} table immediately to pre-fetch four possible LPS intervals and the delay of 64-to-1 multiplexer can be eliminated thus. As a result, the main critical path of the second bin decision is a 4-to-1 multiplexer and an adder. Finally, the value of the second bin is chosen by the most significant bin of O_{LPS} in the first bin part. Note that the updating of R, O, and CM in the second bin decision scheme is not depicted due to it is similar to the one in the first bin decision scheme. In addition, since the computation of renormalization of R and O is occurred in parallel with the computation of $\mathrm{O_{LPS}}$, the delay of R and O renormalization does not affect the decoding performance due to the critical path is on OLPS. With the proposed mathematical transform method, the critical path delay of TSBAD engine is further improved by 28% (from 3.14 ns to 2.26 ns) compared to the traditional TSBAD engine.

TABLE X
CABAC DECODER IMPLEMENTATION RESULT COMPARISONS OF DIFFERENT DESIGNS

Specifications	Proposed	Lin [14]	Chen [17]	Yang [16]
Technology	UMC 90 nm	UMC 90 nm	0.13 μm	TSMC 0.18 μm
Max. frequency	264 MHz	222 MHz	238 MHz	140 MHz
CM memory implementation	Dual-port SRAM (179.36 bytes) +register (222.25 bytes)	Registers (size not available)	Two two-port SRAMs (total 1033 bytes)	Single-port SRAM (441 bytes)
Gate count	51 267[b] (with CM memory)	82 445 (with CM memory)	43 600 (w/o CM memory)	76 333 (with CM memory)
Average bin/cycle	1.69 (at 57.96 Mb/s) 1.84 (at 129.9 Mb/s)	1.96	1.32	0.71 (at 4 Mb/s) 0.86 (at 60 Mb/s)
Throughput[a] (Mbins/s)	446.2 (at 57.96 Mb/s) 485.76 (at 129.9 Mb/s)	435.1	314.2	99.4 (at 4 Mb/s) 120.4 (at 60 Mb/s)

[a]Throughput = (maximum frequency) * (average bin/cycle).
[b]SE parser and hybrid CM memory included.

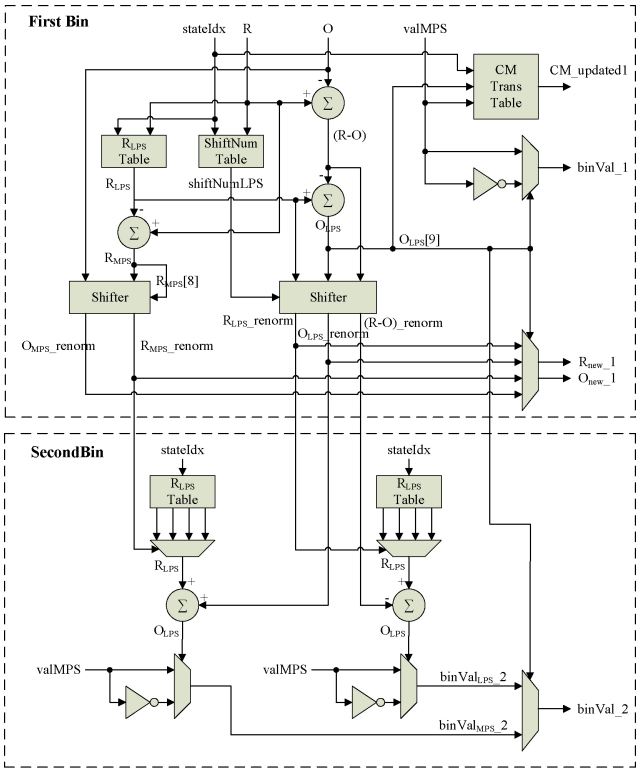

Fig. 11. Detail architecture of proposed two-symbol arithmetic decoding engine.

V. IMPLEMENTATION RESULTS AND PERFORMANCE COMPARISONS

Tables VIII and IX show the decoding performance of the proposed architecture with different QP and frame structure. All the test sequences generated by H.264/AVC reference software have the resolution of HD 1920×1080, the 4:2:0 color format, and the frame rate of 30 ps. With the prediction-based mechanism, the RTL simulation result shows that the proposed design can decode 1.69 bins per cycle on average and over 1.8 bins per cycle in the high bit-rate coding. The average processing time for an macroblock is 175.6 cycles.

The proposed CABAC hardware architecture was synthesized with UMC 90 nm CMOS technology. By applying the mathematical transform method, the proposed architecture can efficiently reduce the critical path delay and allow the maximum working frequency to be about 264 MHz. Table X summarizes the synthesis results and shows the performance comparisons with previous works. Although Lin's design [14] can decode nearly 2 bins per cycle, all register based architecture imposes heavy hardware cost on gate count and its control of SE parser is much more complicated. Chen's work [17] achieved low hardware cost by using a CM memory reallocation strategy. However, it sacrifices the utilization of two-bin arithmetic decoding engine, which only decodes 1.32 bins per cycle on average. With the intrinsic limit of using single-port SRAM, the decoding speed of Yang's design [16] was under 1 bin per cycle. Note that inverse zig-zag scan, inverse field scan, and inverse transform for both 4×4 blocks and 8×8 blocks are also included in his design, and therefore the gate count is higher than [17]. With the proposed prediction-based two-stage pipelining architecture, the data throughput of our design can achieve 485.76 Mbins/s in the high bit-rate coding and 446.2 Mbins/s on average, which outperforms other existing designs. In addition, Tables XI and XII further show the required working frequencies of different levels from the maximum MBs per second and maximum bitrate view point, respectively. The working frequencies listed in Table XI are calculated by the maximum MBs per second multiplies the average processing cycles per MB of our proposed CABAC

TABLE XI
WORKING FREQUENCY FOR DIFFERENT LEVEL FROM THE MAXIMUM MBS PER SECOND VIEW POINT

Level	Max. MBs/Frame	Max. MBs/s	Working Frequency
4	8192	245 760	44 MHz
4.1	8192	245 760	44 MHz
4.2	8704	522 240	92 MHz
5	22 080	589 824	104 MHz
5.1	36 864	983 040	173 MHz

TABLE XII
WORKING FREQUENCY OF LEVEL 5.1 AT 240 MB/S BITRATE CONSTRAINT

Sequence	Total Decoding Cycles	Working Frequency
Blue_sky	176673435.00	176.67 MHz
Riverbed	215280593.00	215.28 MHz
Rush_hour	157131331.00	157.13 MHz
Station2	167354374.00	167.35 MHz
Sunflower	162134918.00	162.13 MHz
Tractor	171589184.00	171.59 MHz
Average	175027305.83	175.03 MHz

decoder. For example, for Level 5.1, the working frequency can be calculated by 983 040 (Max. MBs/s) × 175.6 (average processing cycles per MB) = 173 MHz. From the maximum bitrate view point, since the maximum supported bitrate specified in Level 5.1 is 240 Mb/s, we use several full HD video sequences to generate the bitstreams with the bitrate of 240 Mb/s. In our simulation, 30 frames (30 f/s frame rate) have been encoded to generate bitstreams and the generated bitstreams are decoded by our CABAC decoder. From Table XII, the average required working frequency to support Level 5.1 real-time decoding is 175.03 MHz. As a result, our design can easily achieve Level 5.1 real-time decoding since 264 MHz working frequency is supported by our CABAC decoder. The total gate count of the proposed CABAC decoder is 51.26 k, which includes the hybrid CM memory. Note that the line buffer requirements used to store neighboring information for context selection are not included in our design due to the neighboring information can be obtained from other functional blocks such inter, intra, or deblocking filter in the whole video decoder design [19]. Compared with the all register based architecture, hardware cost saving of 48.6% is achieved.

VI. Conclusion

This paper proposed a prediction-based two-stage pipelining CABAC decoder architecture design for H.264/AVC to accelerate the CABAC decoding procedure without stall for most cases. Based on several advanced techniques proposed in this paper, the SE switching overheads, structure hazards caused by CM reading and writing, and critical path delay of TSBAD engine can be significantly improved. Synthesis results show that the proposed architecture is fast enough to support Level 5.1 real-time decoding with relative lower working frequency.

References

[1] *Advanced Video Coding for Generic Audiovisual Services*, ITU-T Rec. H.264, Mar. 2010.

[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and 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] D. Marpe, H. Schwarz, and 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.

[4] Y. Yi and I. C. Park, "High-speed H.264/AVC CABAC decoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 4, pp. 490–494, Apr. 2007.

[5] L. Bingbo, Z. Ding, F. Jian, W. Lianghao, and Z. Ming, "A high-performance VLSI architecture for CABAC decoding in H.264/AVC," in *Proc. Int. Conf. ASIC*, Oct. 2007, pp. 790–793.

[6] B. Shi, W. Zheng, H.-S. Lee, D.-X. Li, and M. Zhang, "Pipelined architecture design of H.264/AVC CABAC real-time decoding," in *Proc. IEEE ICCSC*, May 2008, pp. 492–496.

[7] P. Zhang, "Fast CABAC decoding architecture," *Electron. Lett.*, vol. 44, no. 24, pp. 1394–1395, Nov. 2008.

[8] Y. H. Liao, G. L. Li, and T. S. Chang, "A high throughput VLSI design with hybrid memory architecture for H.264/AVC CABAC decoder," in *Proc. IEEE ISCAS*, May 2010, pp. 2007–2010.

[9] W. Son and I. C. Park, "Prediction-based real-time CABAC decoder for high definition H.264/AVC," in *Proc. IEEE ISCAS*, May 2008, pp. 33–36.

[10] Y. T. Chang, "A novel pipeline architecture for H.264/AVC CABAC decoder," in *Proc. IEEE Asia Pacific Conf. Circuit Syst.*, Dec. 2008, pp. 308–311.

[11] W. Yu and Y. He, "A high performance CABAC decoding architecture," *IEEE Trans. Consum. Electron.*, vol. 51, no. 4, pp. 1352–1359, Nov. 2005.

[12] C. H. Kim and I. C. Park, "High speed decoding of context-based adaptive binary arithmetic codes using most probable symbol prediction," in *Proc. IEEE ISCAS*, May 2006, pp. 1707–1710.

[13] M. H. Xu, Y. L. Cheng, F. Ran, and Z. J. Chen, "Optimizing design and FPGA implementation for CABAC decoder," in *Proc. HDP*, Jun. 2007, pp. 1–5.

[14] P. C. Lin, T. D. Chuang, and L. G. Chen, "A branch selection multi-symbol high throughput CABAC decoder architecture for H.264/AVC," in *Proc. IEEE ISCAS*, May 2009, pp. 365–368.

[15] P. Zhang, D. Xie, and W. Gao, "Variable-bin-rate CABAC engine for H.264/AVC high definition real-time decoding," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 17, no. 3, pp. 417–426, Mar. 2009.

[16] Y. C. Yang and J. I. Guo, "High-throughput H.264/AVC high-profile CABAC decoder for HDTV applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 9, pp. 1395–1399, Sep. 2009.

[17] J. W. Chen and Y. L. Lin, "A high-performance hardwired CABAC decoder for ultrahigh resolution video," *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1614–1622, Aug. 2009.

[18] *H.264/AVC JM Reference Software* [Online]. Available: http://iphome.hhi.de/suehring/tml

[19] Y. K. Lin, D. W. Li, C. C. Lin, T. Y. Kuo, S. J. Wu, W. C. Tai, W. C. Chang, and T. S. Chang, "A 242 mW 10 mm2 1080p H.264/AVC high-profile encoder chip," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2008, pp. 314–615.

Yuan-Hsin Liao received the B.S. and M.S. degrees in electronics engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 2008 and 2010, respectively.

In 2010, he joined PixArt Imagine, Inc., Hsinchu, Taiwan. His current research interests include video processing, computer vision, and intellectual property and system-on-a-chip design.

Gwo-Long Li (S'09) received the B.S. degree from the Department of Computer Science and Information Engineering, Shu-Te University, Kaohsiung, Taiwan, in 2004, and the M.S. degree from the Department of Electrical Engineering, National Dong-Hwa University, Hualien, Taiwan, in 2006. He is currently working toward the Ph.D. degree from the Department of Electronics Engineering, National Chiao-Tung University, Hsinchu, Taiwan.

His current research interests include the video signal processing and its very large scale integration architecture design.

Mr. Li received the Excellent Master Thesis Award from the Institute of Information and Computer Machinery in 2006.

Tian-Sheuan Chang (S'93–M'06–SM'07) received the B.S., M.S., and Ph.D. degrees in electronic engineering from National Chiao-Tung University (NCTU), Hsinchu, Taiwan, in 1993, 1995, and 1999, respectively.

He is currently an Associate Professor with the Department of Electronics Engineering, NCTU. From 2000 to 2004, he was a Deputy Manager with Global Unichip Corporation, Hsinchu. His current research interests include (silicon) intellectual property and system-on-a-chip design, very large scale integration signal processing, and computer architecture.