Chapter 2
Review of CAVLC, Arithmetic Coding, and CABAC

2.1 Introduction of CAVLC

In the Baseline and Extended profiles of H.264/AVC, except the fixed-length coding, two VLC techniques are supported including: CAVLC for quantized transform residues and Exp-Golomb coding for other syntax elements [Wiegand03]. In the previous standards, entropy coding of residues is based on the forward zig-zag scanned run-length coding and fixed variable-length coding. The H.264/AVC standard adopts backward the zig-zag scanned run-length coding with adaptive VLC.

After intra/inter prediction and 4×4 transform and quantization of prediction residuals, CAVLC [Bjøntegaard02] is used to encode the quantized transformed coefficients, and *Universal Variable Length Coding* (UVLC) is used to encode other types of SEs.

The motivations of using CAVLC for SEs of transform coefficients are as follows: (1) Context adaptive coding can provide better coding efficiency over the whole quality range of the standard than UVLC technique; and (2) Separation of the coding of Run and Level, due to possession of weak correlation, after zig-zag scan can achieve better adaptivity, coding efficiency, and lower the demand on memory than that of Run-Level pair coding of UVLC.

After transformed and zig-zag scanned, the non-zero coefficients are often sequences of ±1. The CAVLC coder indicates the number of high frequency ±1 coefficients by using trailing 1s (*T1s*). For non-zero coefficients, coding efficiency can be improved by using different VLC tables.

An example of CAVLC encoding of one 4×4 block is illustrated in Fig. 2.1. After zig-zag scan, the reordered coefficients are coded as sequence of SEs, including *coeff_token*, *sign_T1s*, 2 values for *level*, *total_zeros*, and 4 values for *run_before*.

These 5 types of SEs will be discussed as follows.

1. *coeff_token* is used to code both the total number of non-zero coefficients (*total_coeff*) and the number of trailing ones (*T1*s). Note that the maximum T1s allowed is 3. Any extra 1-valued coefficient is treated as normal non-zero coefficient. In this case *coeff_token* = 0000100
2. *sign of T1*s is used to code the sign bit of each T1 in reverse zig-zag scanned order. *sign_of_T1* = 011

X. Tian et al., *Entropy Coders of the H.264/AVC Standard*, Signals and Communication Technology, DOI 10.1007/978-3-642-14703-6_2,

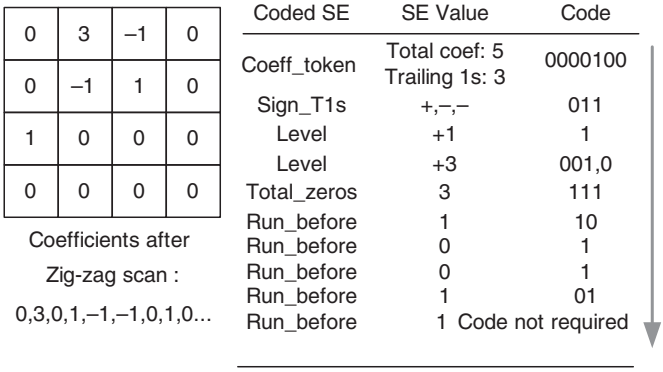

Fig. 2.1 CAVLC encoding of one transform coefficient block [Richardson03]

3. *level* (in sign and magnitude) is used to code the value of each of the remaining non-zero coefficients in reverse order, with the highest frequency non-zero coefficient being encoded first. *level*(1) = 1 (prefix); and *level*(2) = 001 (prefix), and 0 (suffix)
4. *total_zeros* is used to code the total number of zeros preceding the last non-zero coefficient. *total_zeros* is 3 and coded by "111"
5. *run_before* is used to code the number of successive zeros preceding each non-zero coefficient in reverse zig-zag scanned order. *run_before* is coded with 10, 1, 1, 01.

The resulting encoded bit-stream is 0000100_011_1_001_0_111_10_1_1_01.

As shown in Fig. 2.2, both CAVLC decoding and encoding must be executed in 6 steps to completely process the 5 types of SEs of each coefficient block.

From the implementation viewpoint, the *coeff_token* is often obtained by looking up in a 2-D LUT. The selection of LUT of luminance block is based on the number of non-zero coefficients of the previously coded blocks positioning on top and on the left of the block at issue. The *sign_T1s* are output with 1-bit/sign. The *Level* values of the remaining coefficients are coded using selected 1-D LUTs. Each Level-LUT

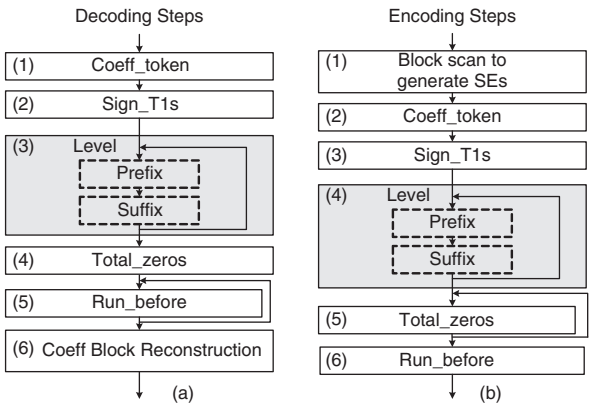

Fig. 2.2 Flowchart for CAVLC codec: (**a**) Decoder and (**b**) Encoder

consists of Exp-Golomb codewords [Golomb66] for levels with small values (associated with high probability), and escapes codewords for levels with large values (low probability). The *Run* values of the block are coded in two steps in LUT: First, the coding of *total_zeros* before the last non-zero coefficient, and second, the coding of *run_before* each non-zero coefficient in the reverse scanned direction.

The coding efficiency of the scheme is high, as the probability is high that one or more low frequency coefficients have no Runs, especially in high bit rate coding. Because the maximum of *total_zeros* is related to non-zero coefficient number, the 1D-LUTs of *total_zeros* are allocated separately for different coefficient number. For *run_before* coding, the number of zeros left to be coded is utilized to select LUT of *zeros_left*. Context adaptivity is introduced to CAVLC coding of all SEs, except *sign_T1s*.

2.2 Pre-CABAC Arithmetic Coders

Compared to the previously reported lossless variable length coders (VLC) [Moffat02] including Elias, Golomb and Rice, Shannon-Fano, and Huffman [Huffman52], the distinct difference of arithmetic coding is that codewords can be represented using fractional number of bits, while in other VLCs, each codeword must occupy integer number of bits. Shannon first mentioned the possibility of such coding technique in 1948 [Shannon48]. Elias explores the idea of successive subdivision of coding interval [Abramson63] in 1960s. Complete scheme of arithmetic coding was proposed by Rissanen [Rissanen76] and Pasco [Pasco76] independently in 1976, in which finite-precision arithmetic coding was implemented. Further research works include hardware-oriented arithmetic coders [Langdon84] of IBM, and software-oriented arithmetic coders by Witten et al.[Witten87], which made it practical in the image and video compression applications.

Multiplication of $\text{Range}_{\text{LPS}}$ (1.38) of subdivision of arithmetic requires high computation and limits the arithmetic coding throughput. Before CABAC, three types of multiplication-free binary arithmetic coders have already been proposed for image compression, which significantly reduce computation complexity of arithmetic coding. The pre-CABAC arithmetic coders include Q-coder [Pennebaker88], and its variants QM coder [Mitchell93] and MQ coder [Taubman00, Taubman02]. QM coder is adopted as the entropy coder in *Joint Bi-level Image Experts Group* (JBIG) standard [JBIG] and *Joint Image Experts Group* (JPEG) standard [JPEG], while MQ coder is adopted in JPEG2000 standard [JPEG2000]. The Q-coder, QM coder, and MQ coder are reviewed as follows.

2.2.1 Q-Coder

Q-coder is an adaptive binary arithmetic coder suitable for both HW and SW implementations. It has simple adapting mechanism which operates on the estimated

probability of coding bin during the coding process. Q-coder avoids the increasing-precision problem using fixed-precision [Rissanen76]. Renormalization is applied after interval subdivision to maintain the size and precision of coding interval. Carry propagation during renormalization is solved by "bit stuffing". When 8 successive bits of "1"s are output, a "0" is stuffed as a carry gap. Multiplication is avoided by using approximated value 1 of Range, because Range is in [0.75, 1.5). Probability estimation is only taken when renormalization is needed for the encoding bin. Computation reduction of interval subdivision and renormalization helps accelerate encoding. However, the precision of arithmetic coding is sacrificed and compression efficiency is degraded. The probability estimation is based on state transition of *Finite-state machine* (FSM), which can be implemented as simple table lookup operation.

2.2.2 QM Coder

QM coder enhances the coding efficiency of Q-coder by incorporation of conditional exchange, and state machine for probability estimation using Bayesian estimation principle for rapid initial learning. The differences between QM coder and Q-coder include [Slattery98]: (a) MQ coder has higher precision with 3 more bits allocated for the coding interval and other intermediate parameters; (b) Q-coder is hardware-based, and QM coder is software-based; (c) Carry propagation is solved at decoder of Q-coder, while it is solved at encoder of QM coder. Bit stuffing is used in Q-coder to prevent carry propagating in the encoder, which is faster than byte-stuffing of QM coder. However, the carry still needs to be solved in the decoder of Q-coder. In comparison, QM coder resolves the carry propagation at encoder by byte stuffing technique, and processing of carry is not needed at decoder.

In the QM coder, adaptive probability estimation is implemented by estimation of next $\text{Range}_{\text{LPS}}$ when renormalization is taken for the coding bin. The estimation is performed by a table lookup operation, and the values of MPS and LPS can be exchanged when the probability of LPS reaches 0.5. For binary arithmetic coding of QM coder, bit assignment of Low and Range is shown in Fig. 2.3. Low register consists of one carry bit, 8 bits of output byte, 3 bits of spacer, and 16 bits of active region. Range register also has 16 bits of active region. During renormalization process, a counter counts the number of valid bits in the output byte section of Low register. When the output byte is full, the byte can be output or recorded as an outstanding byte based on carry bit and whether the output byte is 0xFF. Carry bit of

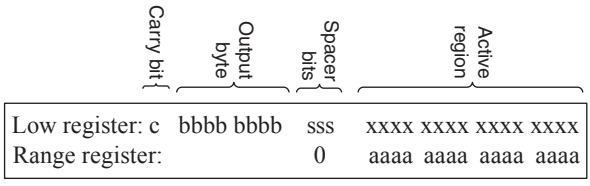

Fig. 2.3 Bit assignment of low and range in QM coder of JPEG

Low can only influence the last byte of bytes in the output buffer. Three spacer bits are allocated in Low between output byte and active region to reduce probability of carry overflow to the output buffer.

2.2.3 MQ Coder

In JPEG2000 [JPEG2000, Taubman02], MQ coder is utilized as the entropy coder of the *Embedded Block Coding with Optimized Truncation* (EBCOT) to encode bit planes of code block of quantized image subbands after *Discrete Wavelet Transform* (DWT) and quantization. MQ coder is closely related to QM coder of JPEG. One distinct difference between the two coders is that the carry is fully resolved in QM coder of JPEG indicating a carry in the Low register will propagate to the nearest bit 0 in the output bits. In comparison, carry propagation in MQ coder is restricted by the mechanism that one bit of 0 is stuffed when the output byte is 0xFF. Any carry from the following renormalization operations cannot influence previous output bytes. Compared to the renormalization and output bit packing mechanism of Q-coder and QM coder, bit stuffing of MQ coder is more efficient in bandwidth and processing time as only one redundant bit stuffing is needed when byte 0xFF is output, and the counter for outstanding bytes is not necessary. ASIC design [Pastuszak05] is reported to accelerate the coding throughput of MQ coder and EBCOT.

2.3 CABAC of H.264/AVC

Although Q-coder, MQ coder, and QM coder are also binary arithmetic coders with statistics adaptivity features, the *Context-based Adaptive Binary Arithmetic Coder* (CABAC) – proposed by Marpe et al. in 2001 [Marpe01a, Marpe01b, Marpe01c] – is adopted as the entropy coding tool for the Main profile and higher profiles of the H.264/AVC standard. Before CABAC, LUT-based VLC are generally utilized for the entropy coding stage in the hybrid block-based video coding standards including H.263, MPEG-2, MPEG-4 Part 2. The weakness of VLCs is that coding event with probability larger than 0.5 cannot be efficiently represented, and the coding procedure is not adaptive to the actual symbol statistics as the values of LUTs are fixed [Marpe03a]. The only arithmetic coder adopted in video standard is of Annex E of H.263 [ITU-T Rec. H.263], in which coding efficiency of entropy coding is not significantly improved, because it directly uses SEs of VLC for arithmetic coding without redefinition. Before CABAC proposals by [Marpe01a, Marpe01b, Marpe01c] of H.264/AVC, similar arithmetic-coder-based implementations were first investigated and applied to the non-block-based video coding [Marpe99, Heising01], such as DWT.

CABAC [ISO/IEC 14496-10, Marpe03a] is the first successful arithmetic coding scheme deployed in the video coding standard, with significant compression improvement compared to the previous entropy coding tools. As shown in Fig. 2.4,

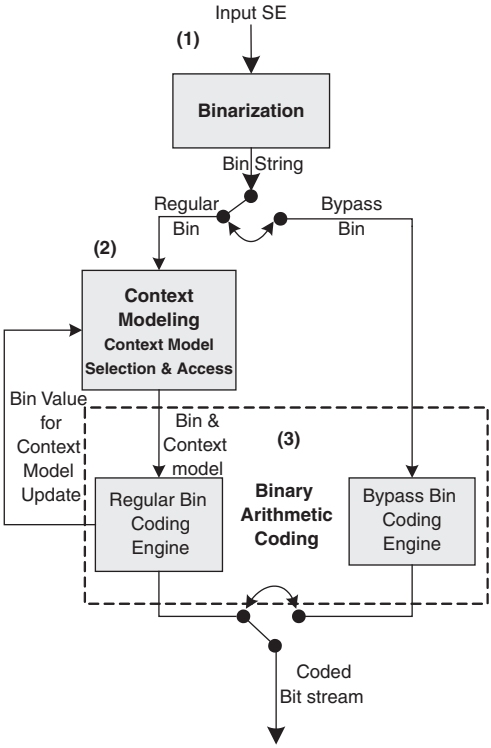

Fig. 2.4 Block diagram of the CABAC encoder of H.264/AVC

CABAC encoding process consists of three elementary *steps*: binarization, context modeling, and binary arithmetic coding (BAC). The input SEs are binarized into bin strings, in which regular bins and bypass bins are encoded separately by the encoding engines of BAC. For regular bin coding, the context model (probability model) of the bin is prepared by the step of context modeling. Details of the three steps are discussed in the following sections.

2.3.1 Binarization

Binarization maps the non-binary valued SE into bin string, which is a sequence of binary decision (bin). Three types of bins are generated in the binarization step: *regular bin*, *bypass bin*, and *terminate bin* for the bins with unequal (variable), equal, or dominant probabilities of value 1 and 0, respectively. Advantages of binarization [Marpe03a] include: (a) the probability of non-binary SE can be represented by the probabilities of individual coding bins, while compression efficiency is not influenced; (b) low-complexity BAC can be utilized; (c) context modeling at sub-symbol (sub-SE) level provides more accurate probability estimation than context modeling at the symbol level, and the alphabet of the encoder is reduced.

Five binarization schemes are used in CABAC: Unary (U), Truncated Unary (TU), kth order Exp-Golomb (EGk), concatenation of the first and third scheme (UEGk), and fixed length binarization (FL). Kth ordered Exp-Golomb binarization (EGk) [Teuhola78] – a derivative of Golomb coding [Golomb66] – is proved to be an optimal prefix-free coding for geometrically distributed sources. EGk codeword consists of prefix and suffix bin strings, with total length of $2l+k+1$ bits. EGk prefix is a Unary codeword, with l bits of 1 and one terminating bit 0. The length l of string of bit 1 is represented as:

$$l = \left\lceil \log_2 \left(\frac{x}{2^k} + 1 \right) \right\rceil \tag{2.1}$$

The length of suffix binary string is equal to $l + k$, and the value of the suffix string is:

$$EGk_{suffix} = x + 2^k - 2^{k+l} \tag{2.2}$$

UEGk is combined binarization scheme of TU and EGk. It is utilized for binarization of SEs of absolute value of residual coefficient level and MVD. TU generates prefix of the bin string, and EGk is adopted to generate the suffix with k set to 0 and 3 for coefficient level and MVD, respectively. TU is simple and it permits fast adaptation of probability of coding symbol. However, it is only beneficial for small SE values. For large SE values, suffix bin string generated by EGk provides a good fit to the probability distribution, and bypass bin coding is also utilized to reduce computation complexity.

2.3.2 Context Modeling

The context modeling step – shown in Fig. 2.4 – implements two sub-functions: context model selection and context model access. The statistics of the coded SEs are utilized to update the probability models (context model) [Wiegand03] of regular bins. For regular bin coding, one context model is chosen, and fetched from a pre-defined set of context models to provide the probability of regular bin to be MPS or LPS, and the context model is updated after bin coding based on bin value. The context index (*CtxIdx*) is calculated to select context model, which is the sum of context offset (*CtxOffset*) and context index increment (*CtxIdxInc*). *CtxOffset* locates the context model set of processed SE, while *CtxIdxInc* selects one context model from the set based on the values of coded bins or coded SEs of neighboring coded blocks.

The idea of multiplication-free arithmetic coding of H.264/AVC is based on the assumption that estimated probability of each context model can be represented by a sufficiently limited set of representative values. In CABAC, the number of the representative values is set to 64 to enable accurate estimation, which is larger than the 30 of Q-coder. Each context model contains a 1-bit tag of MPS value, and a 6-bit probability state index (*pStateIdx*) that addresses one of 64 representative probability

values of LPS from p_0 to p_{63} in the range of [0.01875, 0.5]. The probability values of LPS are derived from (2.3). The ratio of two neighboring probability values is a constant value α, which is approximately equal to 0.949.

$$\begin{aligned} &p_\sigma = \alpha \cdot p_{\sigma-1} \\ &for: \sigma = 1, \ldots, 63, \quad \alpha = \left(\frac{0.01875}{0.5}\right)^{1/63}, and \quad p_0 = 0.5 \end{aligned} \tag{2.3}$$

Probability update of context model is based on the rule in (2.4), in which p_{old} and p_{new} are the probabilities for the bin to be LPS before and after bin coding. If the coding bin is MPS, the probability of LPS decreases by simply multiplying the ratio α, while for the LPS bin, the update probability of MPS is calculated first, and then the probability of LPS is obtained.

$$p_{new} = \begin{cases} \max(\alpha \cdot p_{old}, p_{62}), & if \quad bin = MPS \\ 1 - \alpha \cdot (1 - p_{old}), & if \quad bin = LPS \end{cases} \tag{2.4}$$

By mapping the update probability value of LPS of (2.4) to the closest value in the aforementioned set of representative values, multiplication of probability estimation of CABAC is replaced by simple table lookup for the *pStateIdx* of next probability state according to *pStateIdx* of the current bin, and based on whether it is MPS or LPS. This probability value estimation of context state is actually the function state transition of FSM with 64 predefined states. This type of probability FSM is first utilized in Q-coder, and adopted in QM coder and MQ coder. Compared to Q-coder, QM coder, and MQ coder, the representative LPS probability values need not to be stored in CABAC. Instead, the approximation of the products of coding interval Range and the LPS probability of (1.37) are stored. In order to be more adaptive to the coding context, the values of MPS and LPS can be exchanged when the probabilities of MPS and LPS are equal and the coding bin is LPS.

For particular regular bins of CABAC, multiple context models are allocated for single bin to more precisely represent probabilities of bin in different coding contexts. Four types of context model selection techniques are supported in CABAC, based on (a) neighboring coded SE values of the current SE, (b) values of prior coded bins of SE bin string, (c) position of the to-be-encoded residual coefficient in the scanning path of residual block coefficients, and (d) level values of encoded coefficients of residual block.

2.3.3 Binary Arithmetic Coding (BAC)

BAC performs arithmetic coding of each bin based on bin value, type, and the corresponding context model of the bin. BAC is a recursive procedure of coding interval subdivision and selection, as shown in Fig. 2.5.

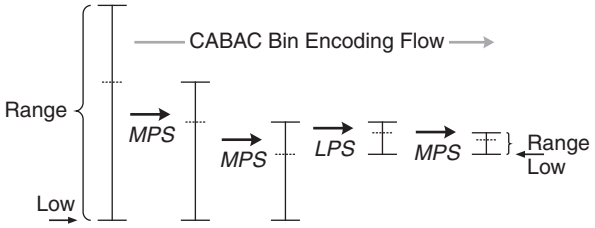

Fig. 2.5 Coding interval subdivision and selection procedure of CABAC

Coding interval subdivision mechanism of CABAC is different from that of QM and MQ coders. In QM and MQ coders, calculation of $\text{Range}_{\text{LPS}}$ of (1.38) is simplified by using the approximated value 1 for Range, and multiplication is eliminated. In comparison, Range is also utilized for $\text{Range}_{\text{LPS}}$ calculation of CABAC. Figure 2.6 shows the reference pseudo C-program of interval subdivision and selection of regular bin, in which the 2 higher-order bits of Range (bit 7 and bit 6) and the index of probability state (*pStateIdx*) of LPS are used to look up for the precalculated product of Range and p_{LPS} of (1.37) in a 2-D LUT. Although the product of LUT is of limited precision, the precision of $\text{Range}_{\text{LPS}}$ calculation, interval subdivision is improved, and computational complexity is reduced in CABAC, compared to those of QM and MQ coders.

```
RangeIdx =  (Range >> 6) & 3; //Range[7:6]
Range_LPS = rangeTableLPS[pStateIdx][RangeIdx];
Range_MPS = Range – Range_LPS;
if(bin == MPS) //bin is MPS
  Range = Range_MPS;
else { //bin is LPS
  Range = Range_LPS;
  Low = Low + Range_MPS;
}
```

Fig. 2.6 Coding interval subdivision and selection of regular bin of CABAC

Because Range and Low of coding interval are represented by finite number of bits (9 bits for Range and 10 bits for Low), it is necessary to renormalize (scale up) the interval to prevent precision degradation, and the upper bits of Low are output as coded bits during renormalization. Coding interval renormalization and bit output of CABAC is based on Witten's algorithm [Witten87], as illustrated in the reference pseudo C-program of Fig. 2.7. The coding interval of (Low, Low + Range) is renormalized when Range is smaller than the threshold value 256 (0x100), which is ¼ of the maximum range of coding interval.

As illustrated in Fig. 2.7, renormalization of Range and Low is an iterative procedure, and the maximum number of iterations is 6, as the smallest possible value of Range is 6. For the processing of carry propagation and output of coding bits, the coded bits of CABAC are not output until it is confirmed that further carry

```
while (Range < 0x100) {
  if (Low >= 0x100) {
     if (Low >= 0x200) {
       //Output bit 0 and following outstanding bits of 1
       PutBit(1);
       Low = Low - 0x200;
     }
     else {  // Low is between 0x100 and 0x200
       Low = Low - 0x100;
       NumOutstandingBits ++;  //Accumulate outstanding bits
    }
  }
  else {
    //Output bit 0 and following outstanding bits of 1
     PutBit(0);
  }
  //scale up Range and Low values by left shift
       Range = Range << 1;
       Low = Low << 1;
}
```

Fig. 2.7 Pseudo C-program of renormalization and bit output of CABAC

propagation will not influence bit values. Figure 2.7 illustrates that only when interval length (Range) is smaller than the threshold 0x100, one bit can be output if the interval is located within the top half [0x200, 0x400) or bottom half [0, 0x200) of the maximum coding range, or an outstanding (OS) bit is accumulated when the interval is within [0x100, 0x300). When a bit of value *X* is output in BAC, the accumulated OS bits are output with the value 1–*X*. Compared to the bit stuffing or byte stuffing schemes of Q-coder, QM coder, and MQ coder, carry propagation is completely solved during renormalization of BAC, and no additional processing of bit stream is needed at CABAC decoder. Moreover, as no bits or bytes are stuffed in the bit stream, the compression efficiency of CABAC is further improved. However, the renormalization illustrated in Fig. 2.8 is a highly sequential operation, and as the number of iterations is variable depending on the selected subinterval Range, it is

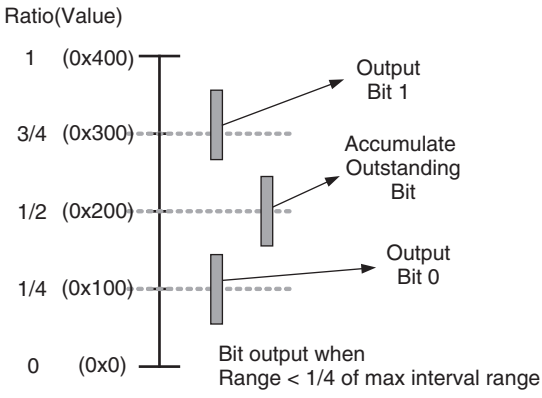

Fig. 2.8 Decision of bit output and accumulation of outstanding (OS)bit

challenging for SW or HW acceleration of renormalization and bit output of BAC. In some situations, long delay may be experienced, when a large number of OS bits are accumulated.

2.3.4 Comparisons of CABAC with Other Entropy Coders

The coding efficiency of CABAC is higher than those of the Q-coder, QM coder, and MQ coder, because of (a) the more precise multiplication of $\text{Range}_{\text{LPS}}$, (b) larger number of probability states for each probability model, and more precise probability estimation of coding bins; and (c) more context models (probability models) deployed for various coding contexts of different types of SEs.

Because of the high computational complexity of CABAC, another entropy coding tool CAVLC [Bjøntegaard02] is deployed in the Baseline profile and Extended profile of H.264/AVC targeting low bit-rate real-time video coding. It offers compression-complexity tradeoff with lower complexity, and lower coding efficiency, compared to CABAC [Marpe03a]. It is employed to encode the quantized transform coefficients of 4×4 residual blocks, while zero-order Exp-Golomb codes [Teuhola78] (EG0) are used for all other types of non-residual SEs. Adaptivity is introduced to CAVLC by allowing switching among multiple VLC tables based on the already processed SEs, and the coding efficiency of CAVLC is better than those of the previous VLC coders which used single VLC table. Instead of coding data pair of run-level as a single SE, run and level of the residual block are encoded separately in CAVLC, so that the inter-symbol redundancy can be more efficiently exploited. However, compression efficiency of CABAC is significantly higher, with typically bit rate reduction of 9–14% in the video quality range of 30–38 dB [Marpe03a], compared to CAVLC and EG0. This is because (a) in CABAC, encoding symbols can be more precisely represented in non-integer number of bits, especially for the symbol with probability higher than 0.5, and (b) CABAC encoder is more adaptive to the non-stationary symbol statistics with efficient context modeling (probability estimation) for the coding bins of all types of SEs.

Since the adoption of CABAC entropy coding in H.264/AVC [Marpe97, Marpe, Heising01, Marpe03a, Mrak03], CABAC is also applied in many applications of image and video processing including motion mode and residual data of 3D dynamic mesh [Muller05], prediction residual in lossless 4D medical image compression [Sanchez08], SEs of 8×8 transform coefficients of AVS coding standard [Zhang07], motion vector coding of scalable video coder [Wu07], parameters of depth and correction vectors in multi-view video coding [Sehoon07]. CABAC is also utilized to encode affine motion vector [Kordasiewicz07], and MVD of 3-D DWT-based subband video encoder [Golwelkar07].