11. Arithmetic coding

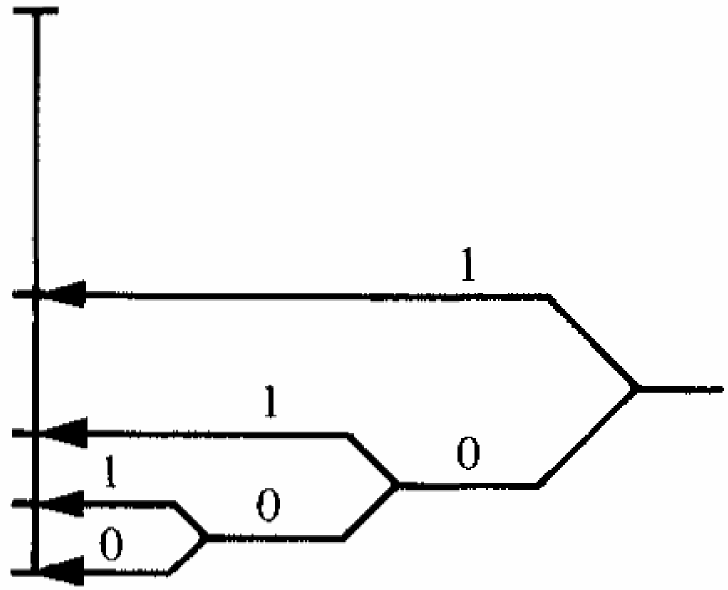
— Case study by Q-Coder and Context-based Adaptive Binary Arithmetic Coding (CABAC)

Yi-Shin Tung
National Taiwan University

Huffman coding

- A way to represent symbols of different probabilities using integral bits, and approximate to its entropy, -log*p(s).*

Symbol	Probability	Code	Binary fraction
z	0.5	1	0.1
y	0.25	01	0.01
x	0.125	001	0.001
w	0.125	000	0.000



Arithmetic coding

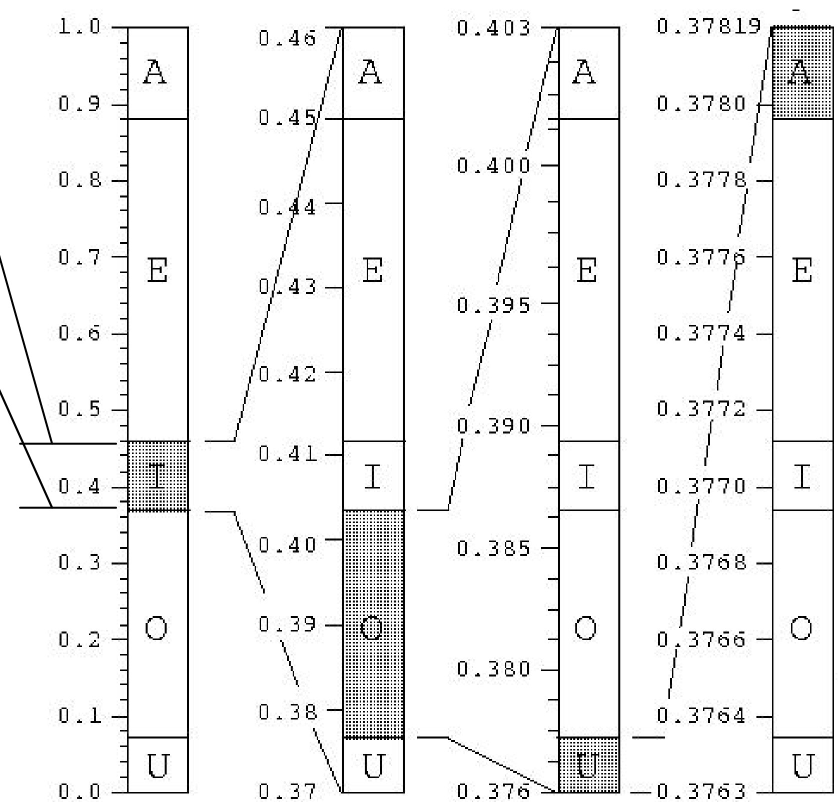
- The problem with this scheme lies in the fact that Huffman codes have to be an integral number of bits long.
- The optimal number of bits to be used for each symbol is -log2(1/p), where p is the probability of a given symbol.
- Thus, if the probability of a character is 1/256, such as would be found in a random byte stream, the optimal number of bits per character is log base 2 of 256, or 8.
- If the probability goes up to 1/2, the optimum number of bits needed to code the character would go down to 1.
- If a statistical method can be developed that can assign a 90% (> 0.5) probability to a given character, the optimal code size would be 0.15 bits. The Huffman coding system would probably assign a 1 bit code to the symbol, which is 6 times longer than is necessary.

Arithmetic coding

$$l^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})F_X(x_n - 1)$$

$$u^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})F_X(x_n)$$

For each symbol to encode, the upper bound $u^{(u)}$ and low bound $l^{(l)}$ of the interval containing the tag for the sequence must be computed.



Problem of arithmetic coding

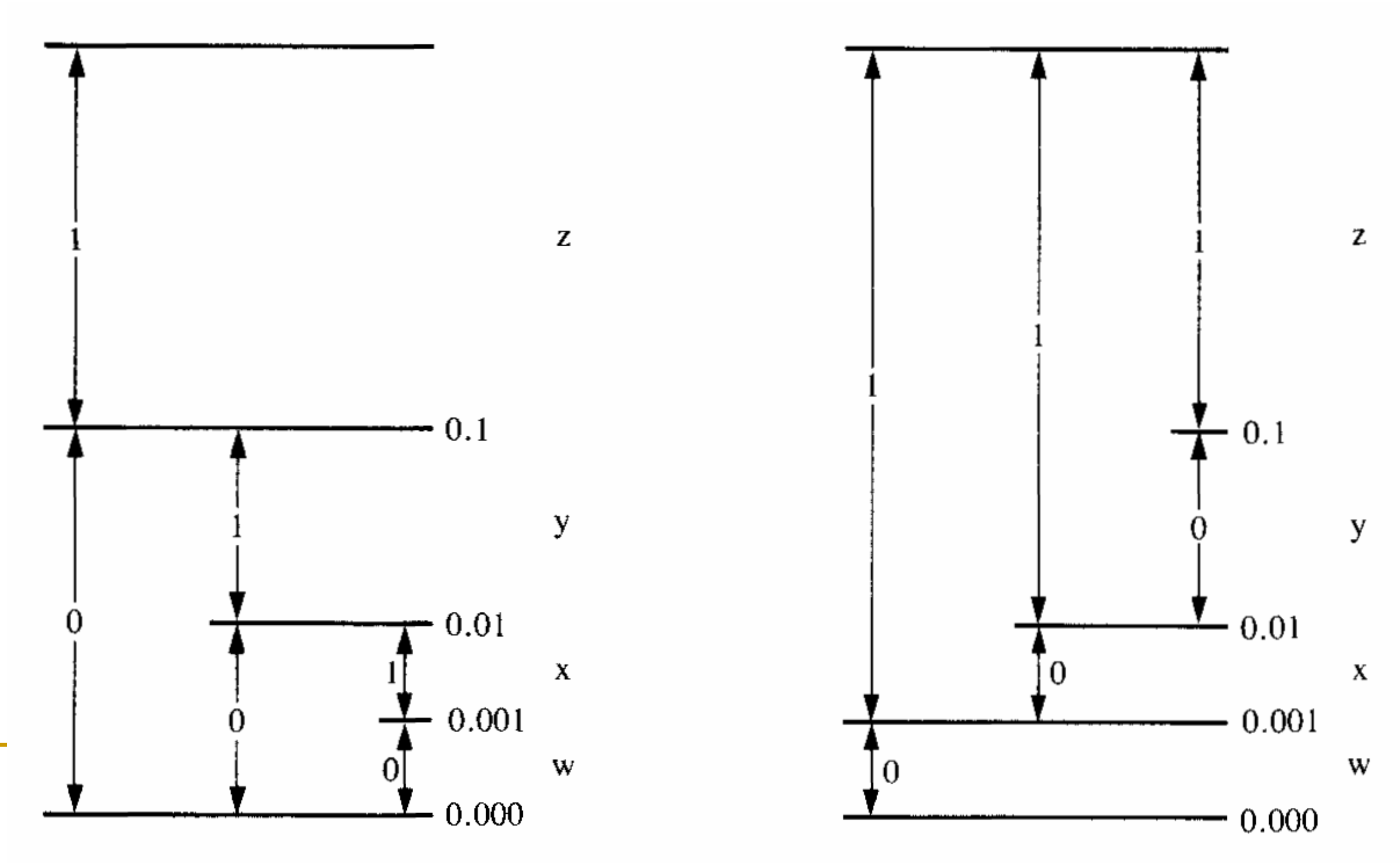
- Calculation precision
- Costly multiplication operation
- Efficient software and hardware implementations
- Effective probability estimation

Binary arithmetic coding

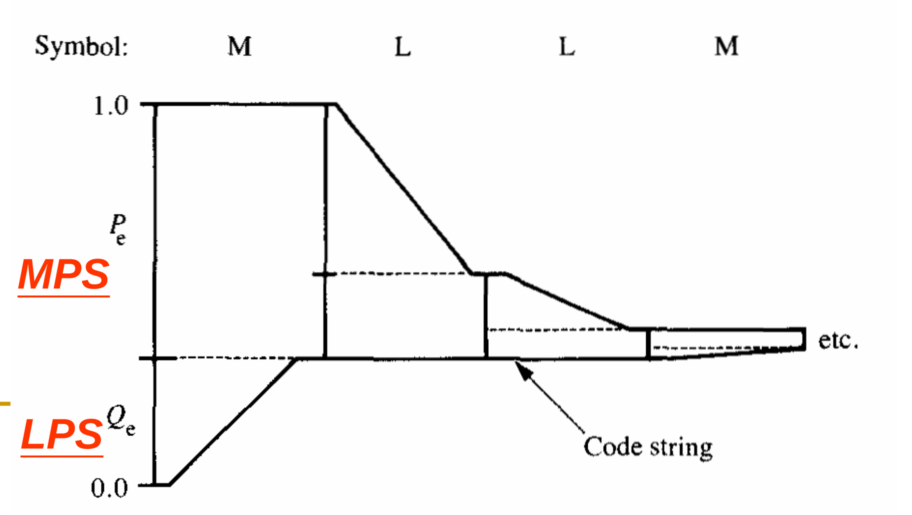
- Any decision selecting one symbol from a set of two or more symbols can be decomposed into a sequence of *binary decisions*.
- Binary arithmetic is based on the principal of recursive interval subdivision.
- Suppose that an estimate of the probability p_{LPS} *in (0,0.5]* of the least probable symbol (LPS) is given and its lower bound L and its range R. Based on this, the given interval is sub-divided into two sub-intervals: $R_{LPS}=R \bullet p_{LPS}$, and the dual interval is $R_{MPS}=R-R_{LPS}$.
- In a practical implementation, the main bottleneck in terms of throughput is the required multiplication operation.
- A significant amount of work has been published aimed at speeding up the required calculation by introducing some approximations of either the range R or of the probability p_{LPS} such that multiplication can be avoided.

Decomposition of binary decisions

- Different decompositions are possible for a fixed probability distribution.
- Huffman coding tree is used as an approximation guide for minimizing the computational burden.

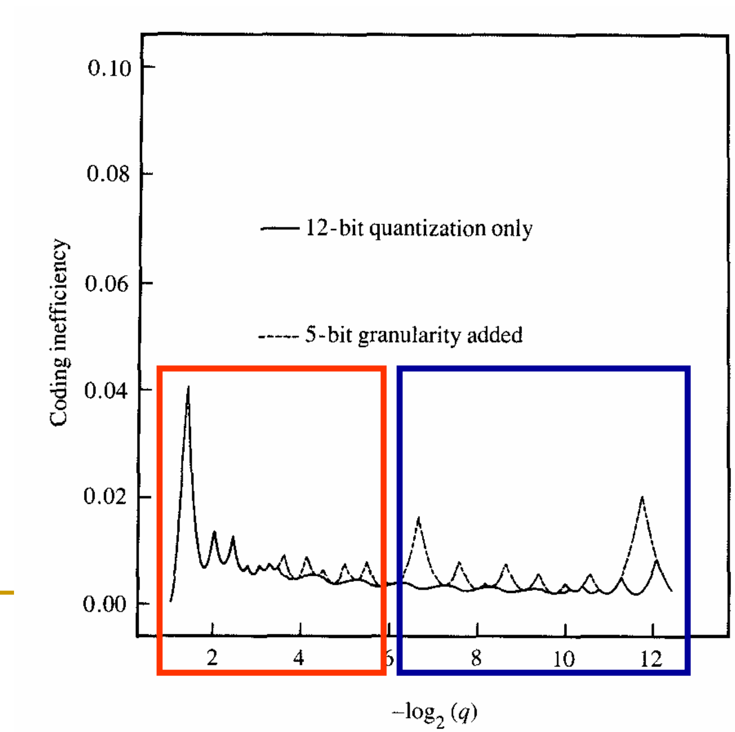


- As coding of each binary decision occurs, the precision of the code string must be sufficient to provide two distinguishable pointers at different symbol subintervals.
- The number of bits, *b*, required to express code string is:
  - $2 \leq 2^b p(s) < 4$
  - $b \leq 2 - \log(p(s))$
- When p(s) goes small, b goes large.
- The translation of the 0 and 1 symbols into MPS and LPS and the subsequent ordering of MPS and LPS subintervals are important for optimal implementation.



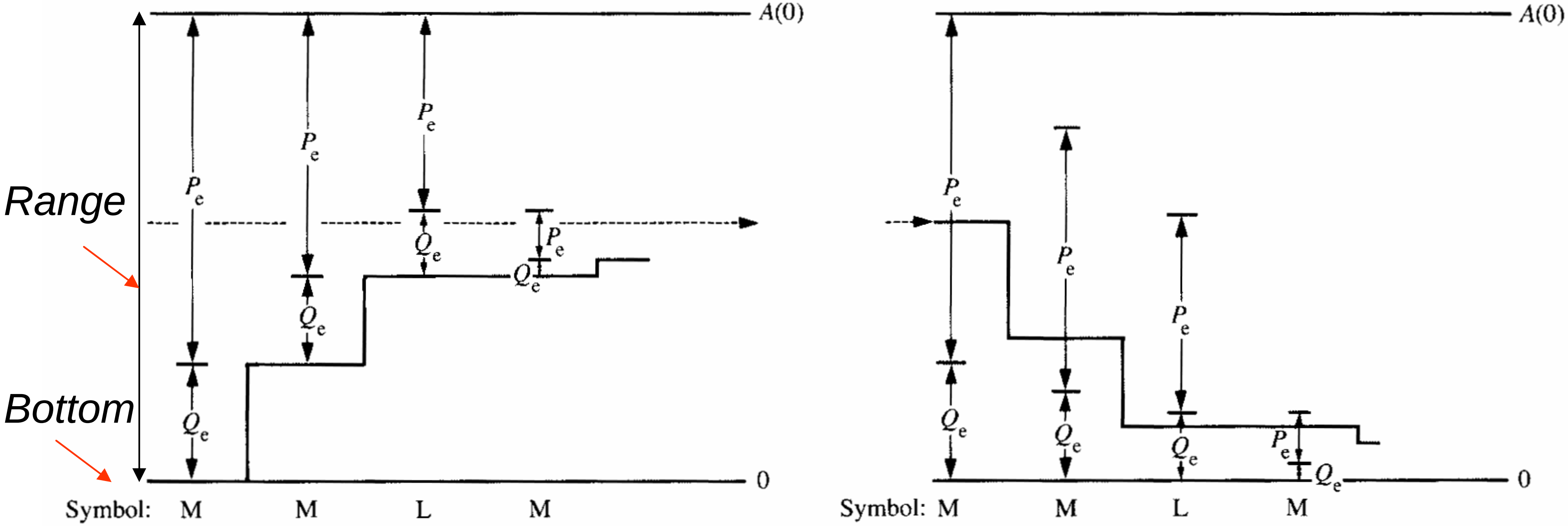
IBM: Q-coder

- Avoid increasing precision problem by using fixed precision arithmetic (12b).
- A renormalization rule must be devised to maintain the interval size within the bound.
- Normalization can be done using shift-left logical operation.
- Multiplication approximation by
  - $AxQ_c \approx Q_c$
  - $AxP_c = Ax(1- Q_c) \approx A - Q_c$
- Coding inefficiency is dominated by
  - Approximation of multiplication
  - Quantization effect



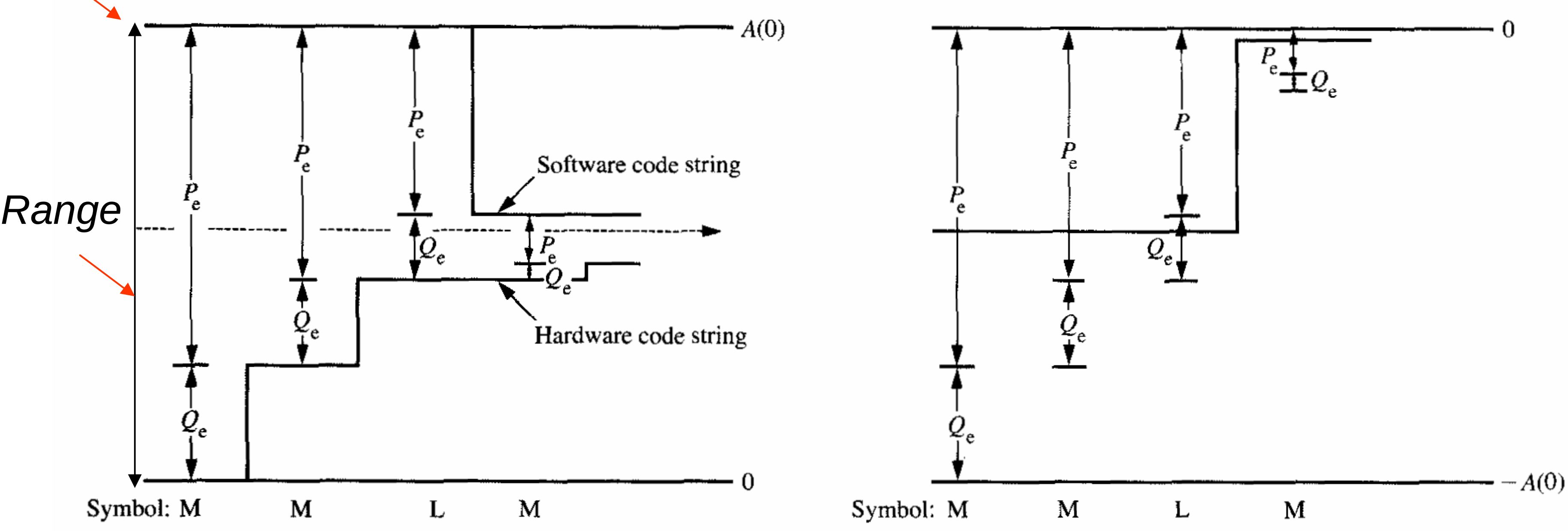
Hardware implementation

- On MPS path, both the current code string and current interval are modified.
- On LPS, only the current interval are changed.
- The extra operations for MPS can be realized in parallel when hardware circuit is developed. (no overhead)



Software implementation

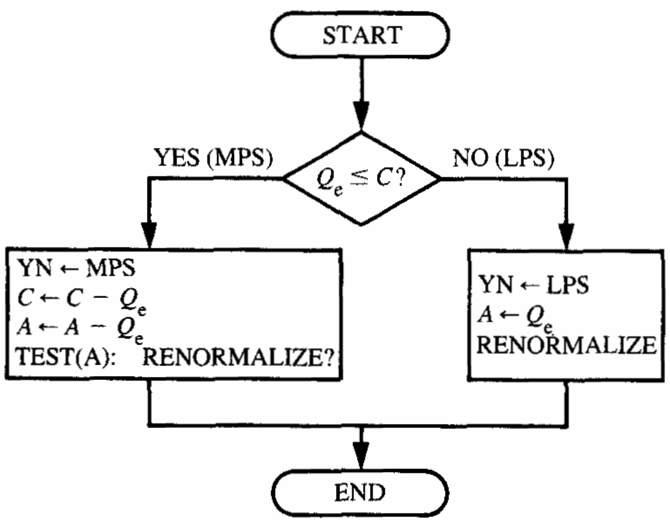
- More operations on MPS are not good for software implementation.
- To eliminate this drawback, one can
  - exchange the order of MPS and LPS,
  - Assign different code-string pointer conventions for hardware and software implementations.



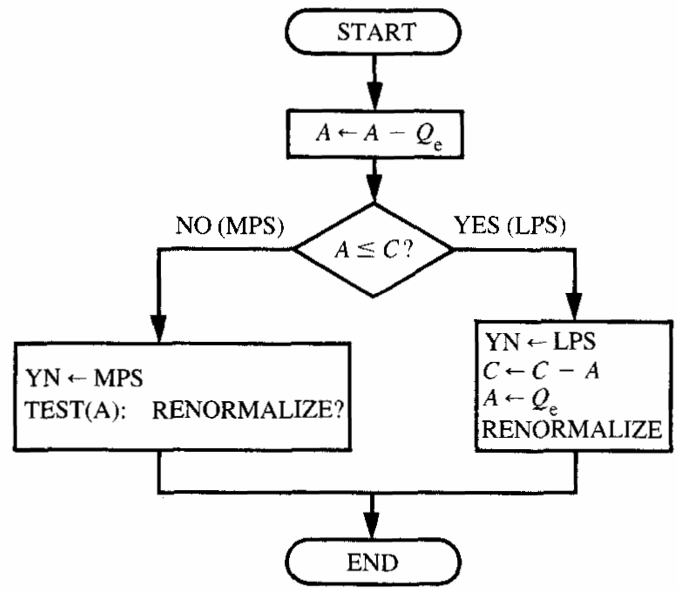
Decoding flowchart

- Different criteria are used for hardware and software implementation, respectively.
  - Parallel processing capability
  - Minimize expected computation

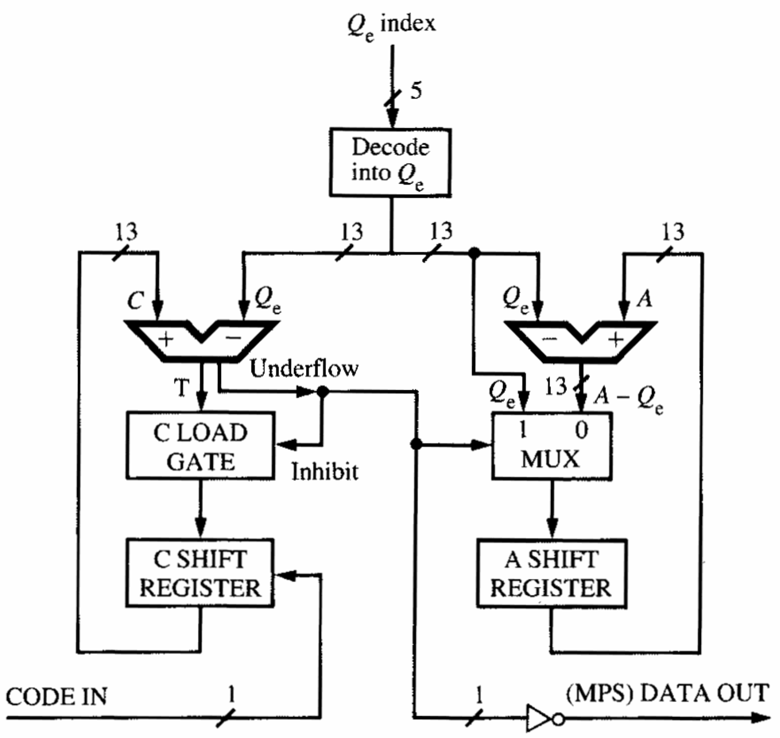
Hardware implementation



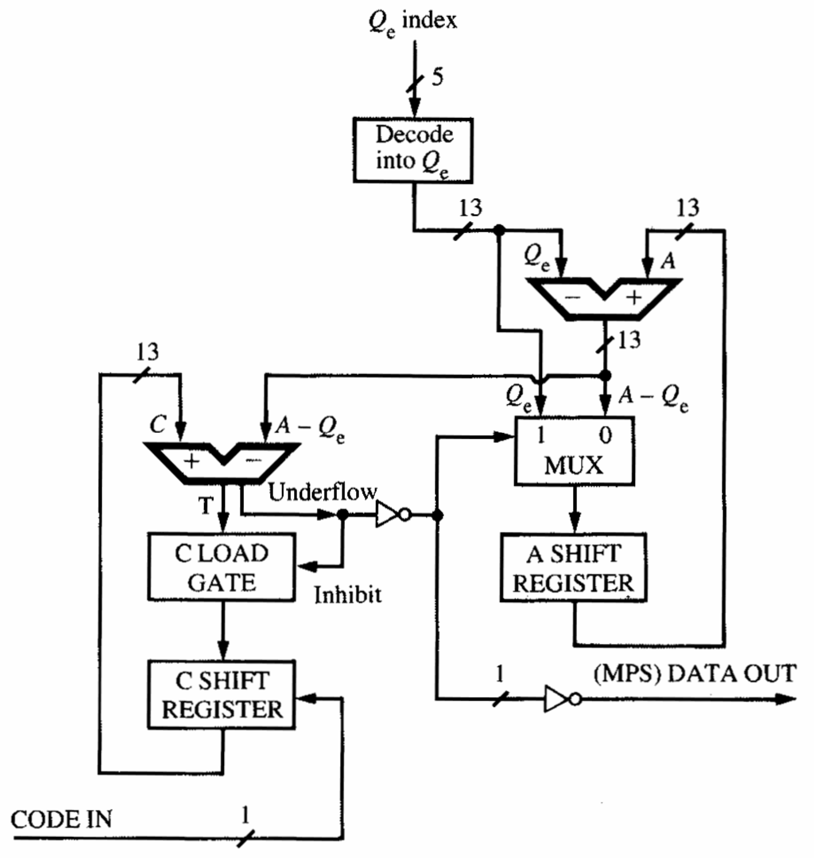
Software implementation



Datapath



Q_e index
5
Decode into Q_e
13
13
13
13
C
Q_e
Q_e
A
Underflow
T
Q_e
13
$A - Q_e$
C LOAD GATE
Inhibit
1 0 MUX
C SHIFT REGISTER
A SHIFT REGISTER
CODE IN
1
1
(MPS) DATA OUT



Q_e index
5
Decode into Q_e
13
13
Q_e
A
13
13
Q_e
$A - Q_e$
C
$A - Q_e$
1 0 MUX
Underflow
T
C LOAD GATE
Inhibit
A SHIFT REGISTER
1
(MPS) DATA OUT
C SHIFT REGISTER
CODE IN
1

Probability estimation

- Adaptive arithmetic coding requires that the probability be re-estimated periodically.
- Estimation only at re-normalization is used in the Q-coder, which is very important for efficient software implementation.
- 60 states are used.
- LPS renormalization v.s. MPS renormalization.

$$N_{\mathrm{mps}} = dA/Q_{\mathrm{e}}\,,$$

$$N_{\mathrm{mps}} = [A - 0.75 + 0.75(dk - 1)]/Q_{\mathrm{e}}\,,$$

$$q = 1/(N_{\mathrm{mps}} + 1) = Q_{\mathrm{e}}\,.$$

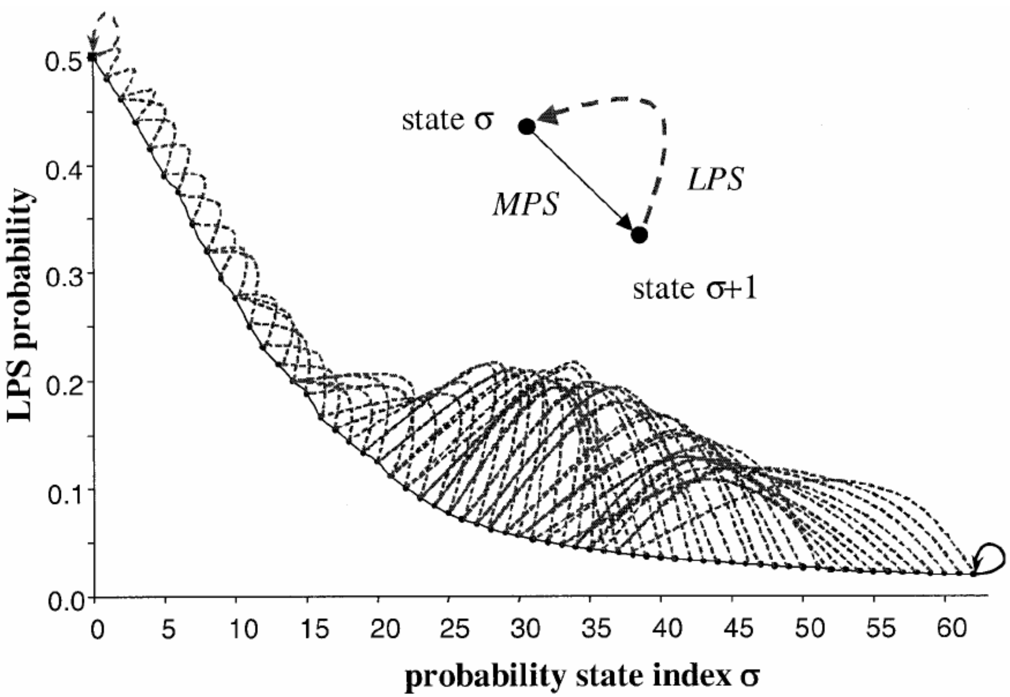
Q_e (hex)	Q_e (decimal)	dk	Q_e (hex)	Q_e (decimal)	dk
X'0AC1'	0.50409	1	X'0181'	0.07050	2
X'0A81'	0.49237	1	X'0121'	0.05292	2
X'0A01'	0.46893	1	X'00E1'	0.04120	2
X'0901'	0.42206	1	X'00A1'	0.02948	2
X'0701'	0.32831	1	X'0071'	0.02069	2
X'0681'	0.30487	1	X'0059'	0.01630	2
X'0601'	0.28143	1	X'0053'	0.01520	2
X'0501'	0.23456	2	X'0027'	0.00714	2
X'0481'	0.21112	2	X'0017'	0.00421	2
X'0441'	0.19940	2	X'0013'	0.00348	3
X'0381'	0.16425	2	X'000B'	0.00201	2
X'0301'	0.14081	2	X'0007'	0.00128	3
X'02C1'	0.12909	2	X'0005'	0.00092	2
X'0281'	0.11737	2	X'0003'	0.00055	3
X'0241'	0.10565	2	X'0001'	0.00018	2

Binary arithmetic coding for H.264

- The Q coder and QM/MQ coders both have their inefficiency. In H.264/AVC, it designed an alternative multiplication-free coder, called modulo coder (M coder), shown to provide a higher throughout than the MQ coder.
- The basic idea of M coder is to project both the legal range $[R_{min}, R_{max})$ of interval width R and the probability range with the LPS onto a small set of representative $Q=\{Q_0,\ldots,Q_{K-1}\}$, $P=\{p_0,\ldots,p_{N-1}\}$. Thus the multiplication on the right-hand side of (3) can be approximated by using a table of $K \times N$ pre-computed values.
- A reasonable size of the corresponding table and a sufficient good approximation was found by using a set Q of $K=4$ quantized range values together with a set P of $M=64$ LPS related probability values.
- Another distinct feature in H.264/AVC, as already mentioned above, is its simplicity bypass coding mode (assumed to be uniformly distributed).

Probability estimation/adaptation

- For CABAC, 64 representative probability values p_σ in [0.01875, 0.5] were derived for the LPS by:
  - $P_\sigma = \alpha * P_{\sigma-1}$ for all $\sigma = 1,\ldots,63$
  - $\alpha = (0.01875 / 0.5)^\wedge(1/63)$ and $p_0 = 0.5$



LPS probability values and transition rules for updating the probability estimation of each state after observing a LPS (dashed lines in left direction) and a MPS (solid lines in right direction).

- Both the chosen scaling factor $\alpha \approx 0.95$ and the cardinality N=64 of the set probabilities represent a good compromise between the desire for fast adaptation (α ➔ 0, small N) and sufficiently stable and accurate estimate (α ➔ 1, large N).
- As a result of this design, each context model in CABAC can be completely determined by two parameters: its current estimate of the LPS probability and its value of MPS β being either 0 or 1.
- Actually, for a given probability state, the update depends on the state index and the value of the encoded symbol identified either as a LPS or a MPS.
- The derivation of the transition rules for the LPS probability is based on the following relation between a given LPS probability p_{old} and its updated counterpart p_{new}:

$$p_{\text{new}} = \begin{cases} \max(\alpha \cdot p_{\text{old}}, p_{62}), & \text{if a MPS occurs} \\ \alpha \cdot p_{\text{old}} + (1 - \alpha), & \text{if a LPS occurs} \end{cases}$$

Table-based binary arithmetic coding

- The internal state of the arithmetic encoding engine is as usual characterized by two quantities: the current interval R and the base L of the current code interval.
- First, the current interval R is approximated by a quantized value $Q(R)$, using an equi-partition of the whole range $2^8 \leq R < 2^9$ into four cells. But instead of using the corresponding representative quantized values Q_0, Q_1, Q_2, and Q_3. $Q(R)$ is only addressed by its quantizer index ρ, e.g. $\rho = (R >> 6)\ \&\ 3$.
- Thus, this index and the probability state index are used as entries in a 2D table TabRangeLPS to determine (approximate) the LPS related sin-interval range R_{LPS}. Here the table TabRangeLPS contains all 64x4 pre-computed product values $p_\sigma \bullet Q_\rho$ for $0 \leq \sigma \leq 63$, and $0 \leq \rho \leq 3$ in 8 bit precision.

CABAC decoding in AVC

- Context handling operation
- Bitsream bit operation
- SE binarization scheme
- Binary arithmetic decoding

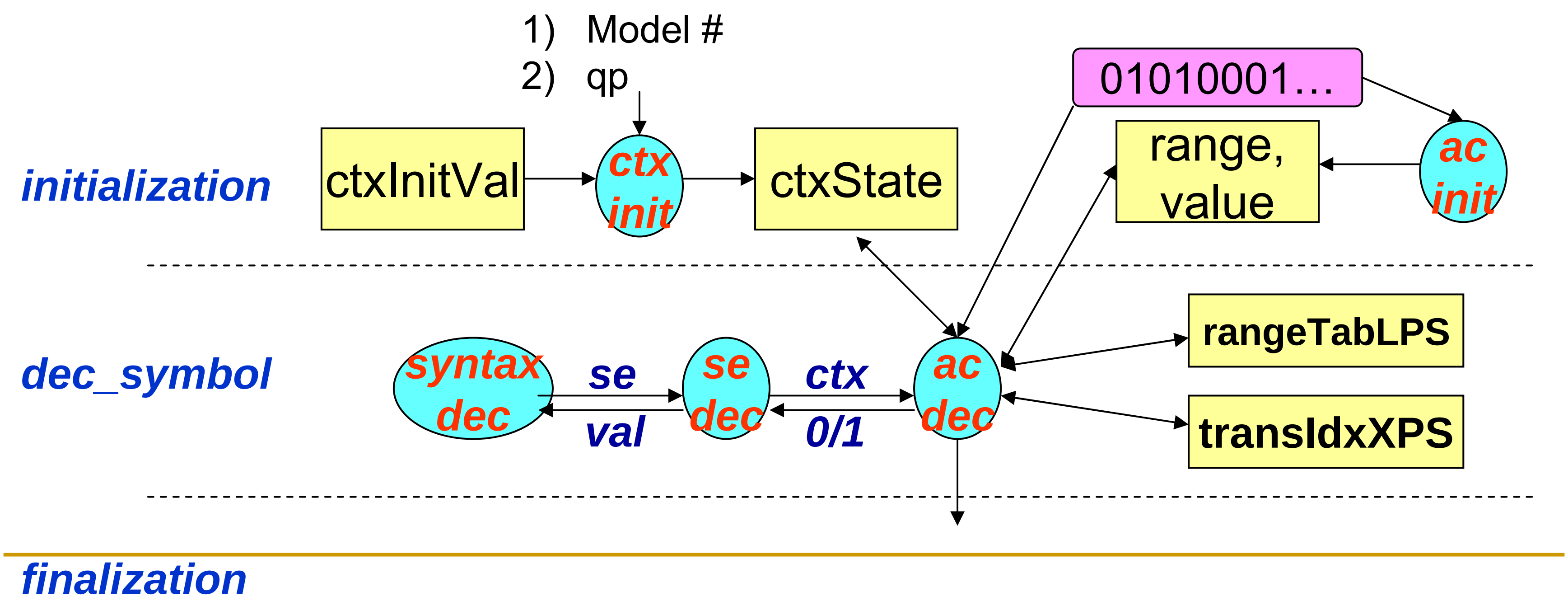
Context initialization

- Contexts are re-initialized at the start of each slice and adaptive automatically and manually.
  - Context model selection (*model_number*), automatically
    - I-Slice has 1 model and P-/B-slices can have 3 models.
  - Coding setting adaptation (*qp*), manually
  - 459 adaptable contexts + 1 static termination context for binary decision.
- Initialization process for each context

```
pstate = ((ini.scaler*qp)>>4) + ini.offset;
pstate = CLIP3(pstate, 1, 126);
ctx.MPS = (pstate >= 64);
ctx.state = MUX(pstate >= 64, pstate – 64, 63 – pstate);
```

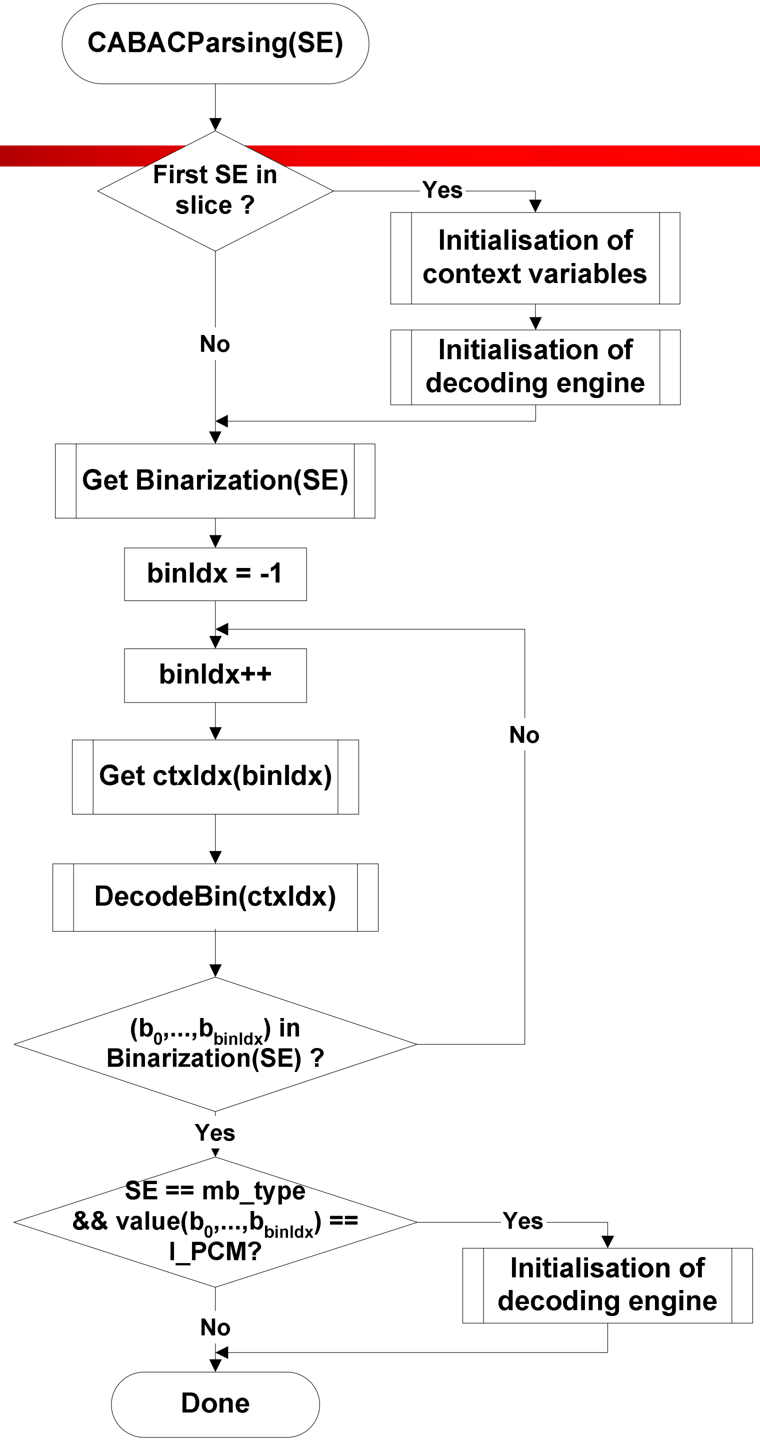
Memory requirement

- range, value (9b x 2)
- rangeTabLPS (const 8b x 64 x 4)
- transIdxLPS/transIdxMPS (const 6b x 64 x 1)
- ctxInitVal (const 8b x 2 x 459 x 3)
- ctxState (7b x 459)



SE decoding

- Syntax element types
- Binarization methods
- Binary Arithmetic coding



Renormalization

- Initialization of decoding engine

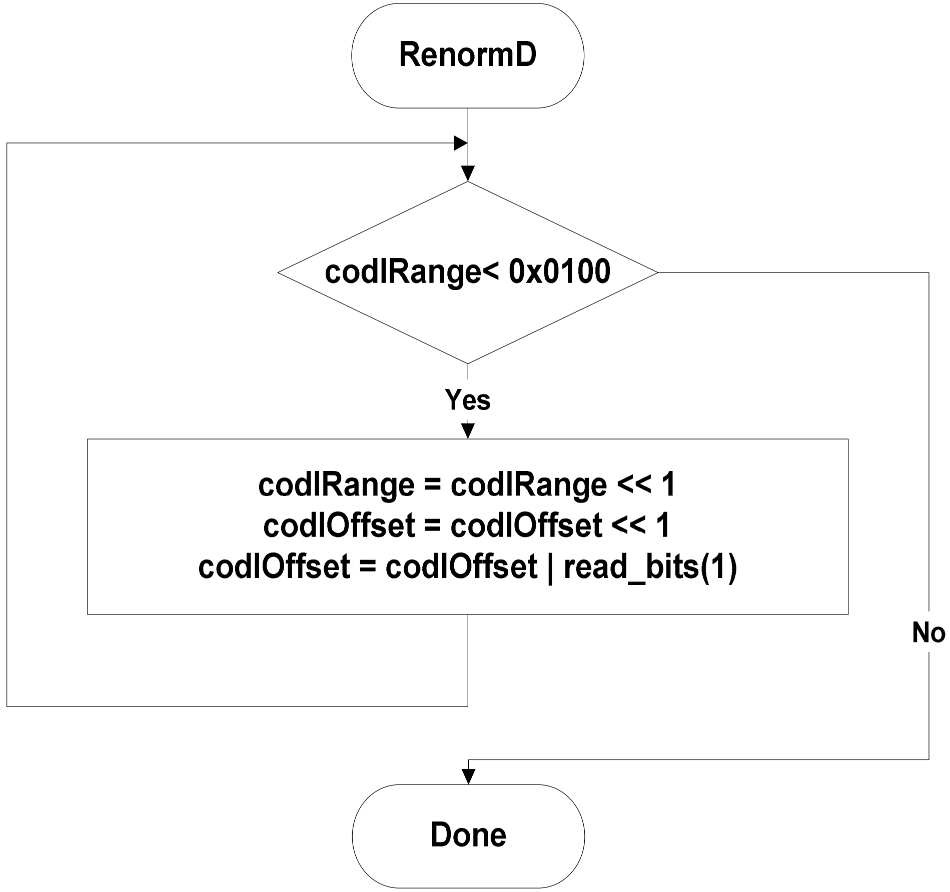
  if (!bytealigned(bs))

      bytealign(bs);

  Dvalue= getbits(bs, 9);

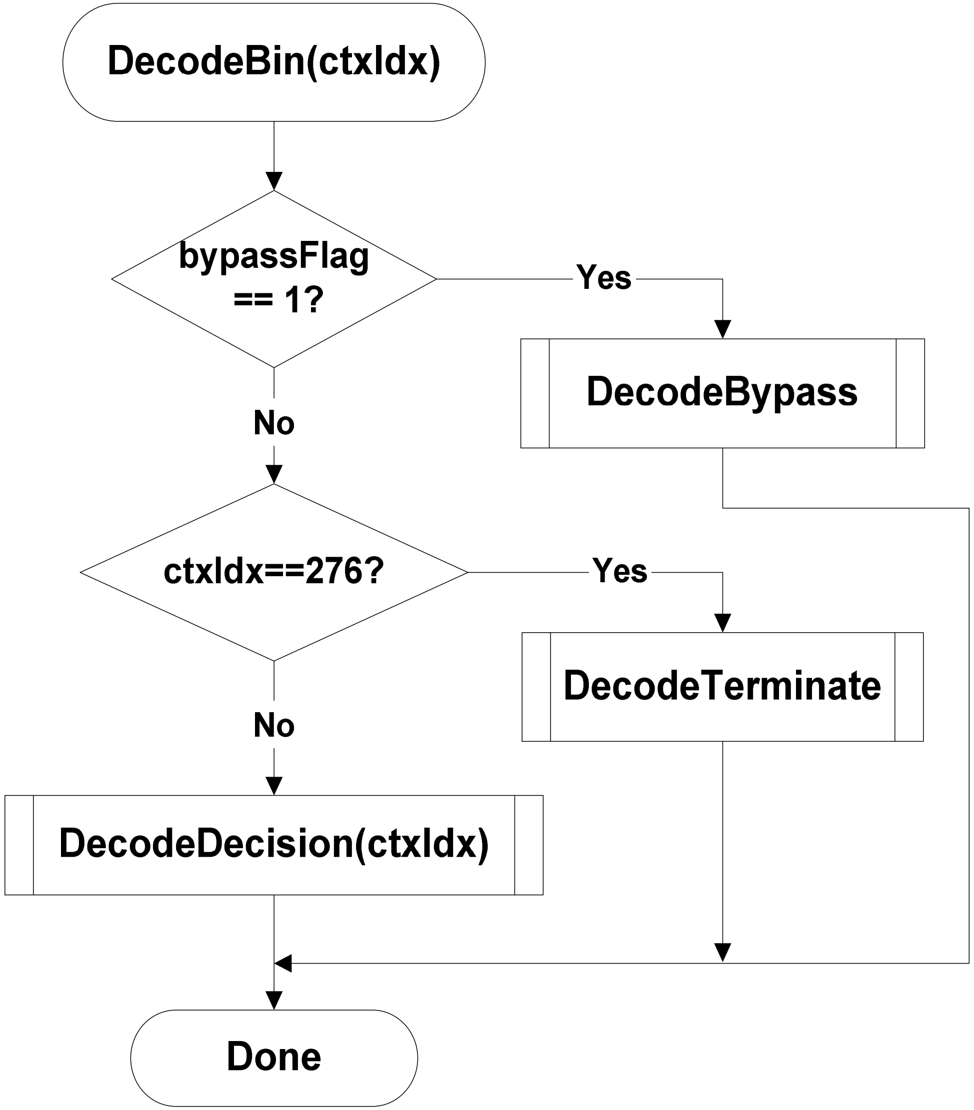
  Drange= 0x01fe;

- Always keep range in [0x100~0x1ff] and value in [0, range)



Decoding flowchart for a binary decision

- Normal mode
  - Adaptive probability
- Bypass mode
  - Equal probability
- Termination mode
  - Extreme probability

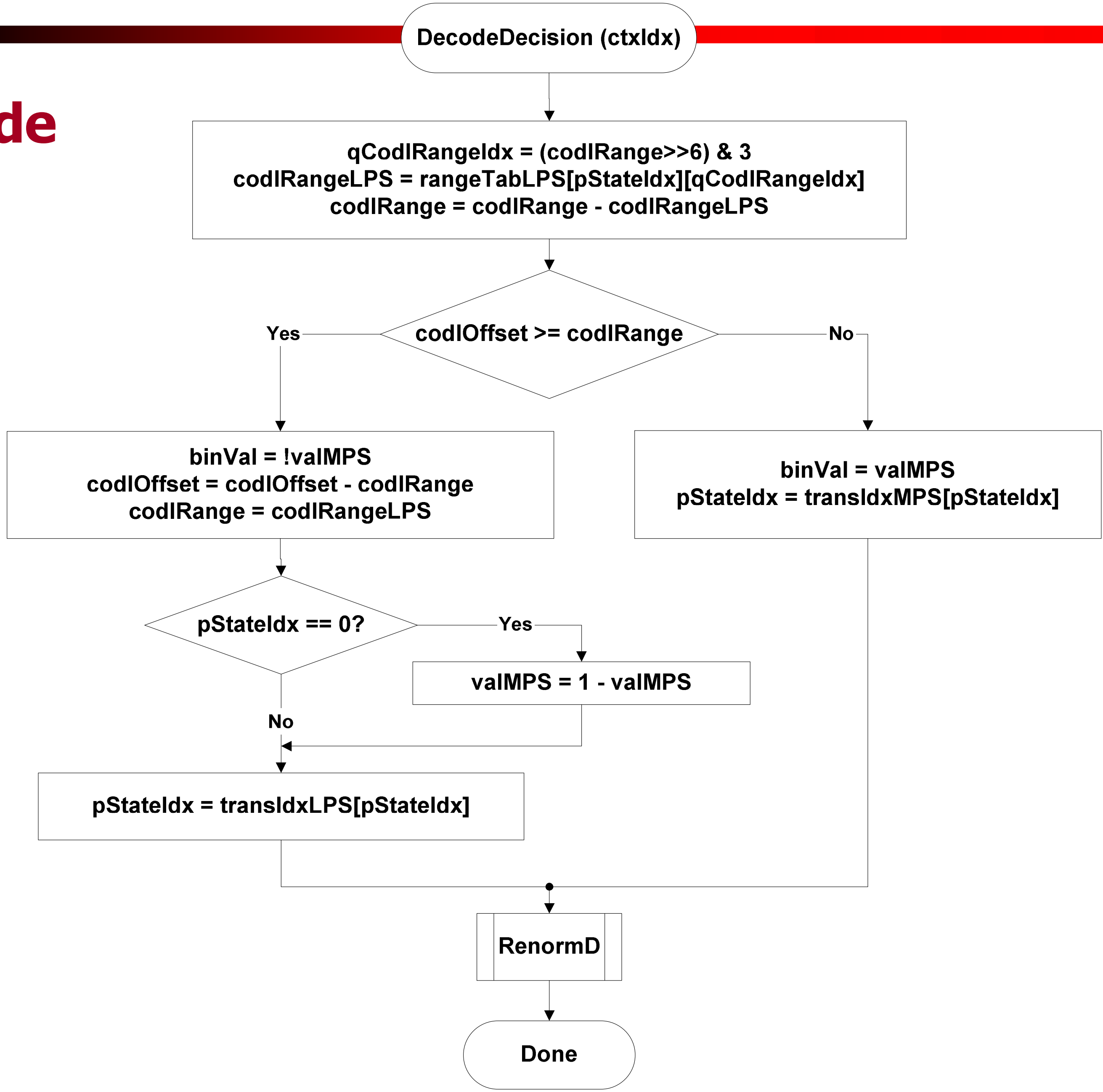


Normal mode

range

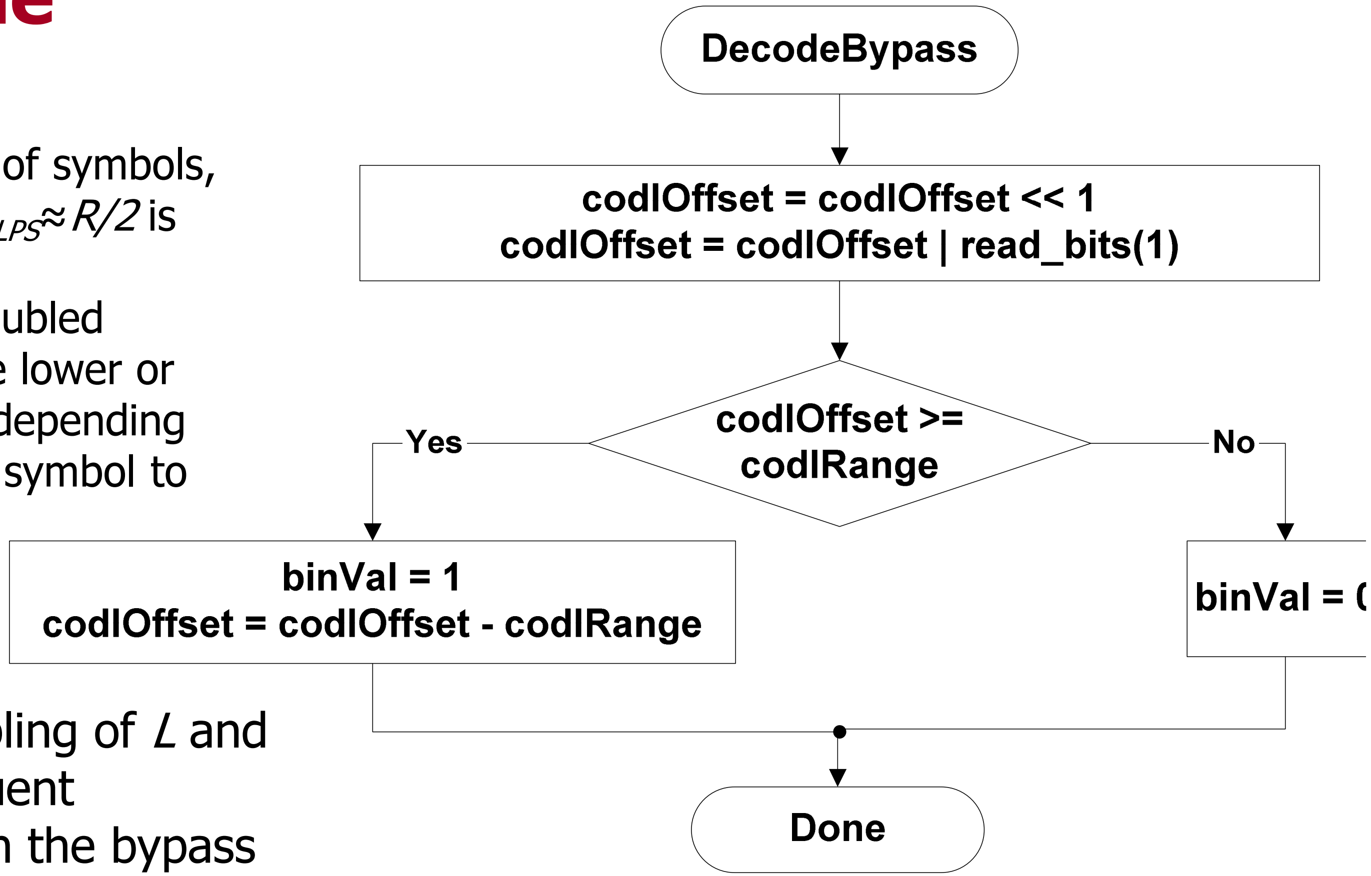
1 xxxx xxxx

idx	avg
00	288
01	352
10	416
11	480



Bypass mode

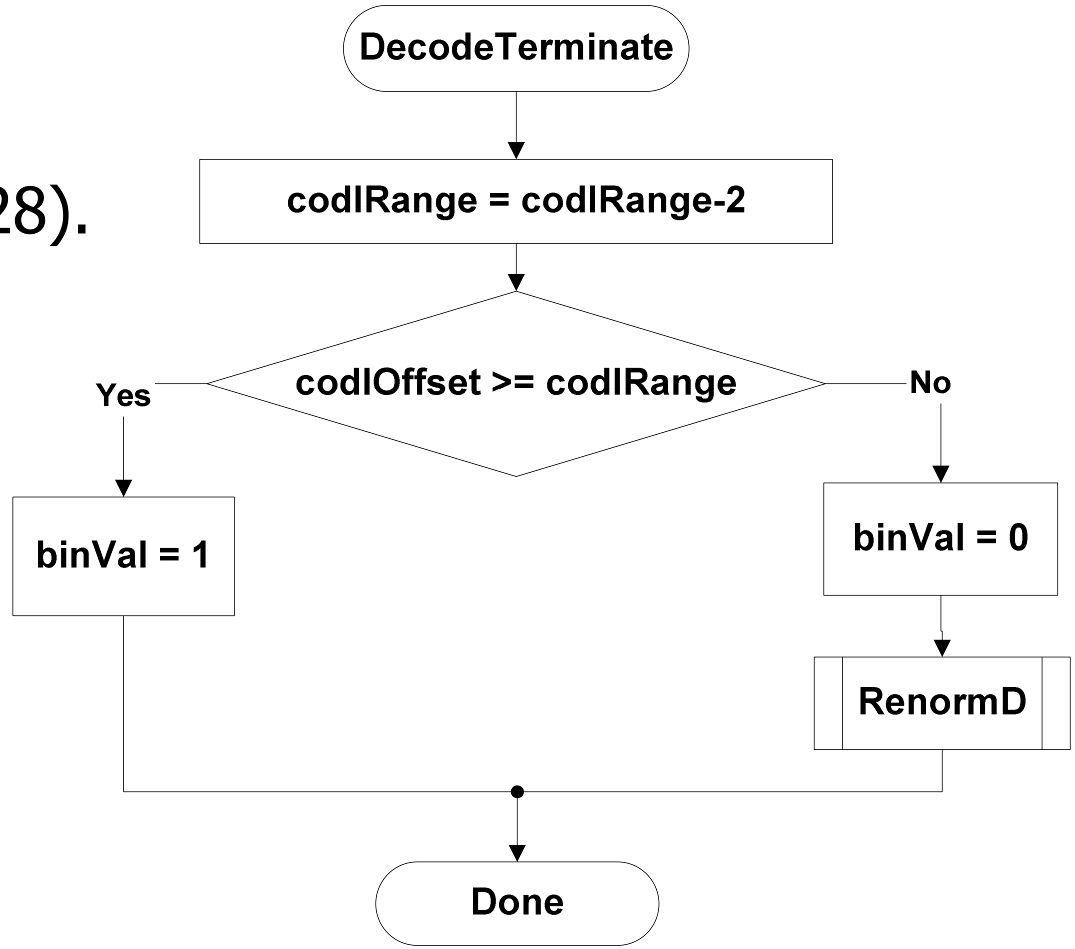
- To speed up the encoding/decoding of symbols, for which $R\text{-}R_{LPS} \approx R_{LPS} \approx R/2$ is assumed to hold.
- The variable L is doubled before choosing the lower or upper sub-interval depending on the value of the symbol to encode (0 or 1).



- In this way, doubling of L and R in the sub-sequent renormalization in the bypass is operated with doubled decision threshold.

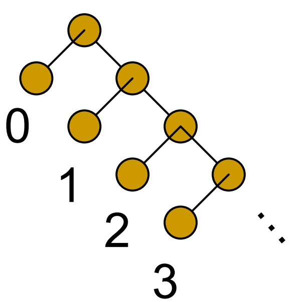
Session termination mode

- Used to indicate end-of-macroblock, PCM macroblock type.
- Probability less than 0.5~1% (1/256~1/128).
- Static probability.

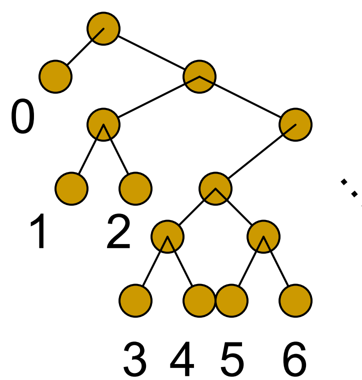


Binarization process

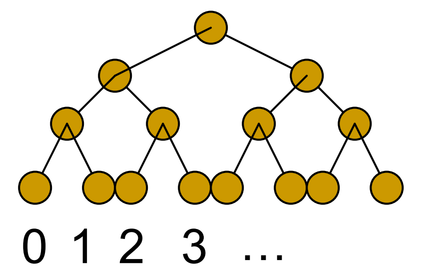
- Binarization types
  - Unary (U)
  - Truncated unary (TU)
  - Exp-golomb (EG)
  - Concatenated unary/k-th order Exp-golomb (UEGk)
  - Fixed length (FL)
  - Customization (combination of others)



[Unary]



[Exp-Golomb]



[Fixed length]

SE coding: MB type, sub-MB type

- In usual, each bin represents a different meaning of coding conditions, e.g. intra/inter, coded/not-code.
- 8 (I-slice), 7 (P-slice), 9 (B-slice) contexts for MB type.
- 3 (P-slice) and 4 (B-slice) contexts for sub-MB type.

Example of binarization for I-MB type

Symbol	Value
0	Intra4x4
1 1	PCM
1 0 ? ?… ??	Intra16x16
1 0 0 ?… ??	… luma NC
1 0 1 ?… ??	… luma Coded
1 0 ? 0 ??	… chroma NC
1 0 ? 10 ??	… chromaDC Code
1 0 ? 11 ??	… chromaDCAC Code
1 0 ? ?… 00	… vertical prediction
1 0 ? ?… 01	… horizontal prediction
1 0 ? ?… 10	… DC prediction
1 0 ? ?… 11	… plane prediction

ctx0~2 ctx3 ctx4~5 ctx6~7
Static ctx

SE coding: ref index and delta qp

- Picture reference index
  - The neighboring ref-indices are used to model zero condition.
  - Separate contexts for different directions.
  - Higher correlation for large values.
- Delta quantizer parameter
  - The last decoded delta-qp is used to model zero condition.
  - Less correlation for larger values.

Symbol	Value
0	0
1 0	1
1 10	2
1 110	3
1 1110	4
1 1…10	5~

ctx0~3 ctx4 ctx5

Unary

Symbol	Value
0	0
1 0	1
1 10	-1
1 110	2
1 1110	-2
1 1…10	±3~

ctx0~1 ctx2 ctx3

SE coding: MV residue

- MV residues in the left and top blocks are used to select active context.

$$absmvd_x = |mvd_{x,up}| + |mvd_{x,left}|$$

$$ctxIdxInc = \begin{cases} 0, & absmvd < 3 \\ 1, & 3 \le absmvd \le 32 \\ 2, & 32 < absmvd \end{cases}$$

- Condition of zero *mvd* is first coded.
- Small *mvd* is of high probability and separate contexts provide higher prediction precision.
- mvd_x and mvd_y are coded by different sets of contexts.
- Totally 14 contexts.

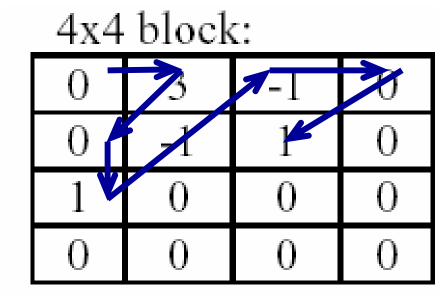
Symbol	Value
0	0
1 0 s	±1
1 10 s	±2
1 110 s	±3
…	±4~±7
1 1111110 s	±8
1 11111111 0 xxx s	±9~ ±16
1 11111111 10x xxx s	±17~ ±32
1 11111111 110xx xxx s	±33~ ±64
…	…

ctx0~2 ctx3~5 ctx6 bypass bypass

UEG3, signedVal=1, uCoff=9

SE coding: significant map

- Use significant bit and last significant bit to indicate locations of nonzero coefficients in the zigzag scan order.
- The last coded significant bit implies the significant status of the last coefficient.



sig-0→(*sig-1*,*last-sig-0*)→
sig-0→(*sig-1*,*last-sig-0*)→
(*sig-1*,*last-sig-0*)→
(*sig-1*,*last-sig-0*)→
sig-0→(*sig-1*,*last-sig-1*)

- Totally 2x ((2x15+2x14+1x3)x2+(15+9))=2x146=292 contexts.
  - Luma4x4 and lumaDC have 15 pairs of (*sig*, *last-sig*)
  - LumaAC and chromaAC have 14 pairs.
  - ChromaDC has 3 pairs.
  - Luma8x8 has 15 *sig* and 9 *last-sig* contexts.

SE coding: coefficient level

- Condition of one is coded by 5 contexts, which represent the number of continuous ones encountered. (n.a., 0, 1, 2, 3)
- Small level is of high probability.
- Magnitude of value is modeled by the number of decoded larger-than-one coefficients. (0, 1, 2, 3, 4)
- Different types of blocks are coded by different sets of contexts.
- Totally 59 contexts for 6 block types.
  - Chroma DC does not have ctx9.

Symbol	Value
0 s	±1
1 0 s	±2
1 10 s	±3
1 110 s	±4
1 11111…10 s	±5~±13
1 $(1)^{12}$1 s	±14
1 $(1)^{12}$1 0 s	±15
1 $(1)^{12}$1 10x s	±16~ ±17
1 $(1)^{12}$1 110xx s	±18~ ±21
1 $(1)^{12}$1 1110xxx s	±22~ ±29
…	…

ctx0~4 ctx5~9 bypass bypass

UEG0, signedVal=0, uCoff=14

SE coding: others

- Mb_skip_flag: 1 bin, 3 contexts for P and B slices, respectively.
- Intra_chroma_pred_mode: 3 bin (TU). 3 contexts for first bin and 1 context for the remainders.
- Prev_intra_luma_pred_mode: 1 bin, 1 context.
- Rem_intra_luma_pred_mode: 3 bin (FL), 1 context.
- Coded_block_pattern: 6 bins (FL). 4 contexts for luma, chromaDC and chromaAC, respectively.
- Coded_block_flag: 1 bin, 4 contexts for each block type except for luma8x8.
- MB_field_coding: 1 bin, 3 contexts.
- Transform_size_8x8: 1 bin, 3 contexts.

Reading assignment

- W.B. Pennebaker, J.L. Mitchell, G.G. Langdon Jr., R.B. Arps, “An overview of the basic principles of the Q-coder adaptive binary arithmetic coder,” Vol 32, No 6, Nov 1998, IBM journal, research development.