VLD in H.264

By C.G Liu

Agenda

- 1 Overview
- 2 Huffman
- 3 Unary Coding (Golomb Coding)
- 4 CAVLC
- 5 CABAC
- 6 VLD in ds2
- 7 Q&A

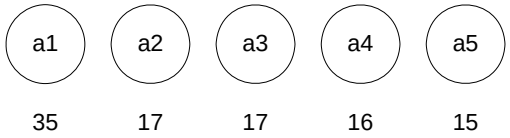
Overview

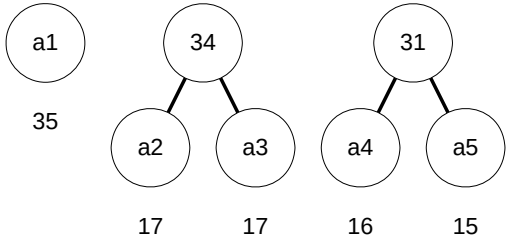
- H262 (MPEG-2) & VC-1
  - Uses static Huffman coding for entropy coding
- H264 (MPEG-4 Part 10)
  - 4 profiles
  - Baseline & Extended – Context Adaptive Variable Length Coding (CAVLC)
  - Main and High – CAVLC or Context Adaptive Binary Arithmetic Coding (CABAC)

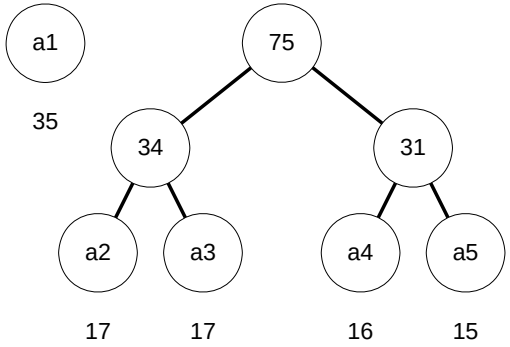
Huffman

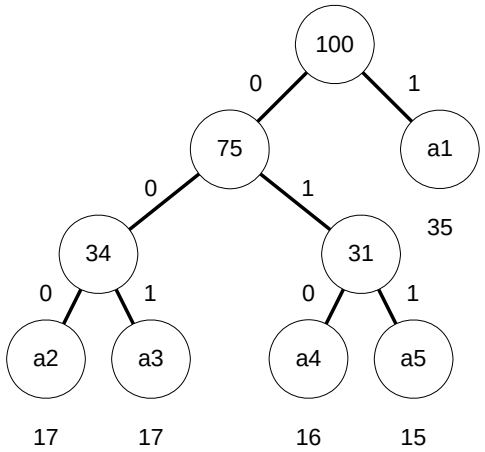
- Huffman encoding
  - Generate statistics of symbols occuring
  - Generate Prefix code (uniquely decodable) based on probability of occurance

Huffman Coding Example









a1	1
a2	000
a3	001
a4	010
a5	011

Huffman Coding

- Good – Simple to encode & decode
  - Can be two pass (Generate Table, Encode)
- MPEG-2 uses static Huffman coding
  - Tables generated after lots of experiments
  - Transmitting new tables not negligible
  - Very close to optimal, but must assign whole bits

In H264

1. Unary Coding (Golomb Coding)
   - Header information
   - 4 Types – Unsigned Unary, Signed Unary, Truncated and Mapped
2. CAVLC-context-adaptive variable length coding
3. CABAC

Unary Coding (Golomb Coding)

- codeNum can be decoded as follows:
  1. Read in M leading zeros followed by 1.
  2. Read M-bit INFO field.
  3. codeNum = 2^M+ INFO – 1

Bit string	codeNum
1	0
0 1 0	1
0 1 1	2
0 0 1 0 0	3
0 0 1 0 1	4
0 0 1 1 0	5
0 0 1 1 1	6
0 0 0 1 0 0 0	7
0 0 0 1 0 0 1	8
0 0 0 1 0 1 0	9
...	...

Unary Coding (Golomb Coding)

- For ue , code_num = k
- For se, code_num = 2|k| (k $\leqslant$ 0) ;code num = 2|k|-1 (k> 0)
- For te, if x >1 te=ue;else te=!read_bit(1)
- For me,

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
0	47	0
1	31	16
2	15	1
3	0	2
4	23	4

CAVLC

- When entropy_coding_mode is 0,for residual_block

Which features are used in CAVLC

- Blocks are typically sparse.(run-level)
- The highest nonzero coefficients after the zig-zag scan are often sequences of ± 1.
- The number of nonzero coefficients in neighbouring blocks is correlated. (nC,6t)
- The level of nonzero coefficients tends to be larger at the start and smaller towards the higher frequencies.(level)

CAVLC Example

4x4 block:

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

Reordered block:
0,3,0,1,-1,-1,0,1,0…

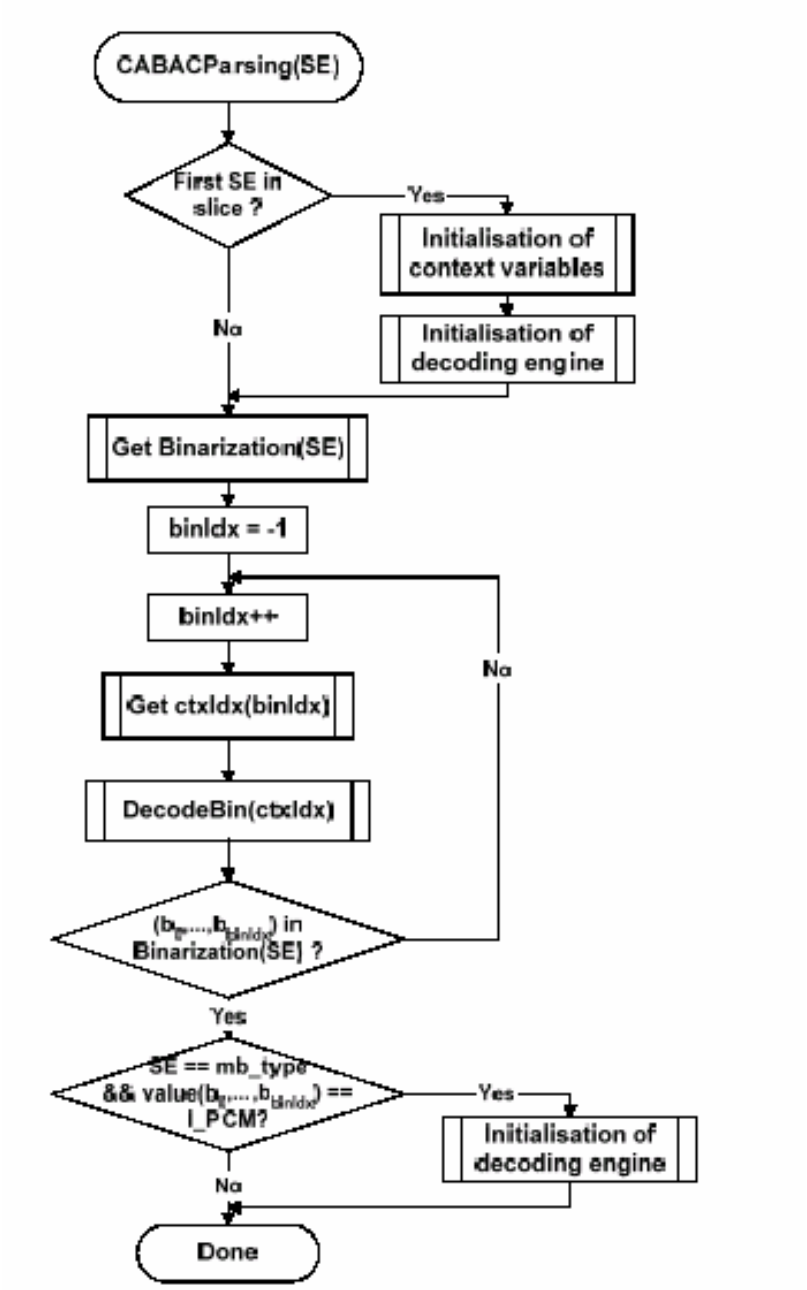
Element	Value	Code
coeff_token	TotalCoeffs=5, T1s=3	0000100
T1 sign (4)	+	0
T1 sign (3)	-	1
T1 sign (2)	-	1
Level (1)	+1 (use Level_VLC0)	1
Level (0)	+3 (use Level_VLC1)	0010
TotalZeros	3	111
run_before(4)	ZerosLeft=3; run_before=1	10
run_before(3)	ZerosLeft=2; run_before=0	1
run_before(2)	ZerosLeft=2; run_before=0	1
run_before(1)	ZerosLeft=2; run_before=1	01
run_before(0)	ZerosLeft=1; run_before=1	No code required; last coefficient.

The transmitted bitstream for this block is 000010001110010111101101 .

CABAC

- When entropy_coding_mode_flag is 1, for residual_block

CABAC Overview



Initialising CABAC Engine

- Need to initialise before first symbol in slice
- Initialise Context
  - Need **sliceQPY**, **cabac_init_idc**
  - Initialise all 460 contexts
  - Output: **pStateIdx[ctx]** , **valMPS[ctx]**
- Initialise Arithmetic Engine
  - codIRange = 0x01FE
  - codIOffset = Read(9)

Binarization process

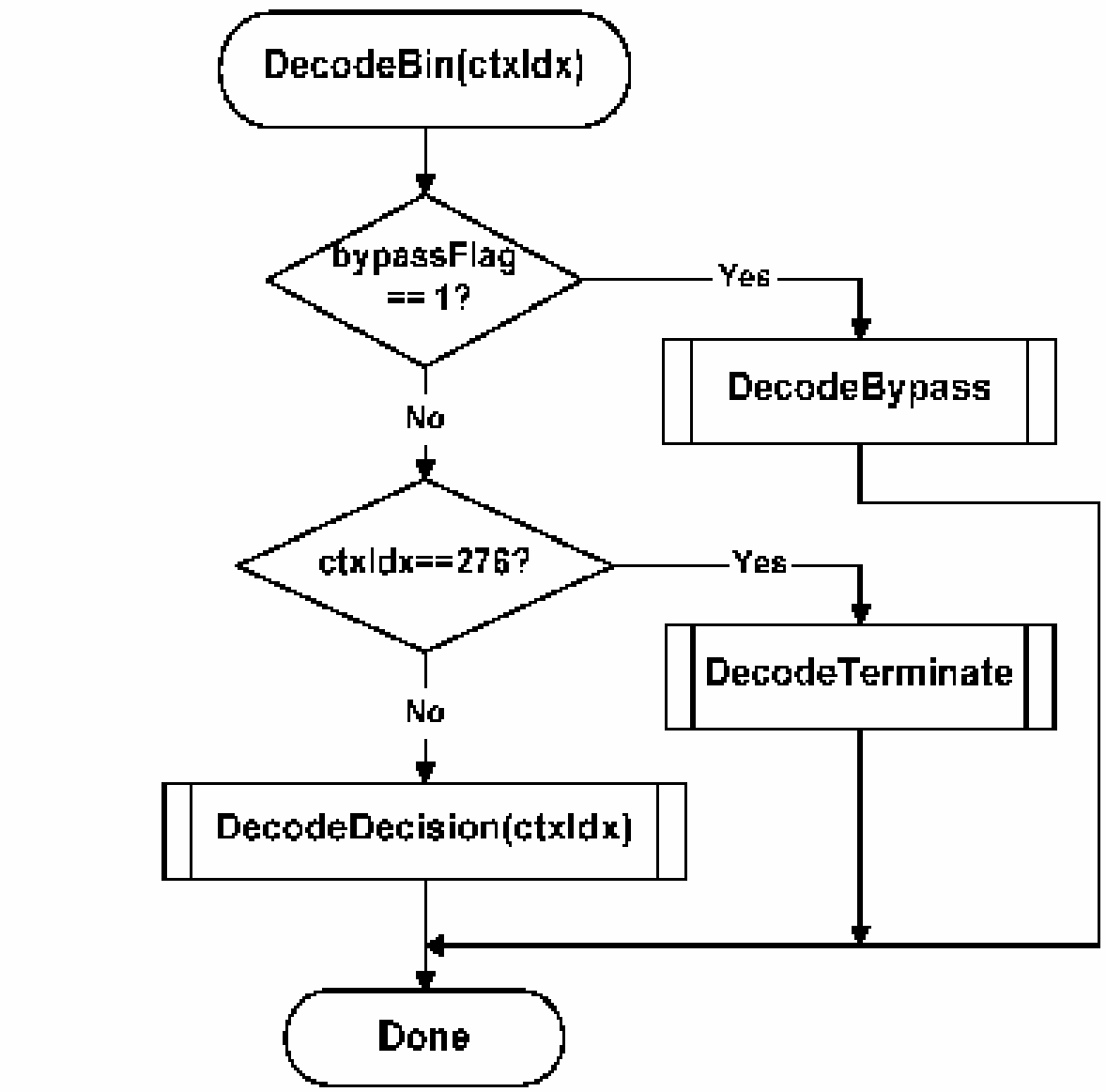
- Given Syntax Element (SE)
- SE has associated binarization method
  - Special (mb_typr,sub_mb_type)
  - Unary
  - Truncated Unary
  - Kth order Exp-Golomb
  - Fixed-length
  - Concatenation(FL+TU,TU+EGk)
- Produce possible *bin string* (codeword)

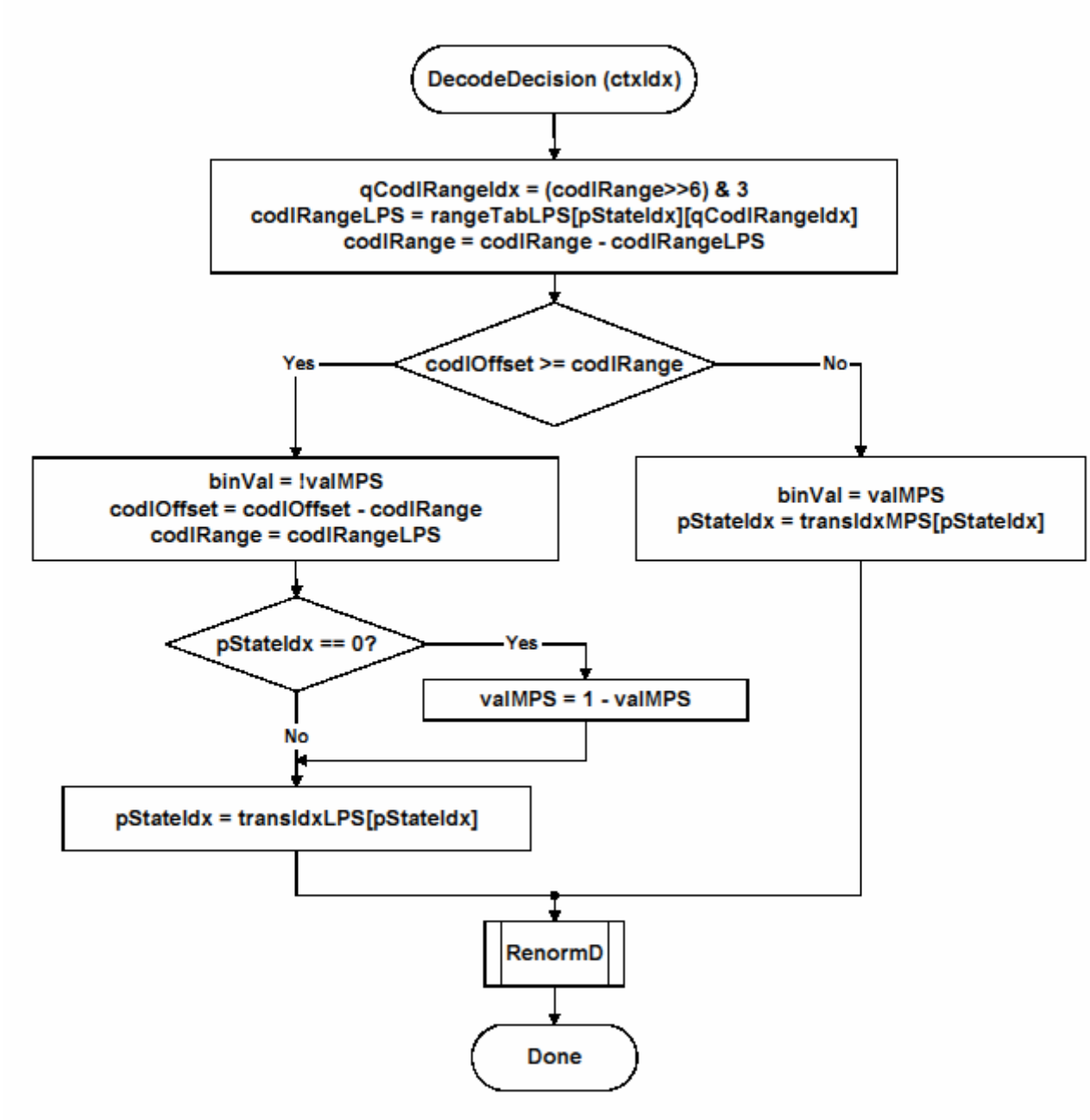
Context modeling

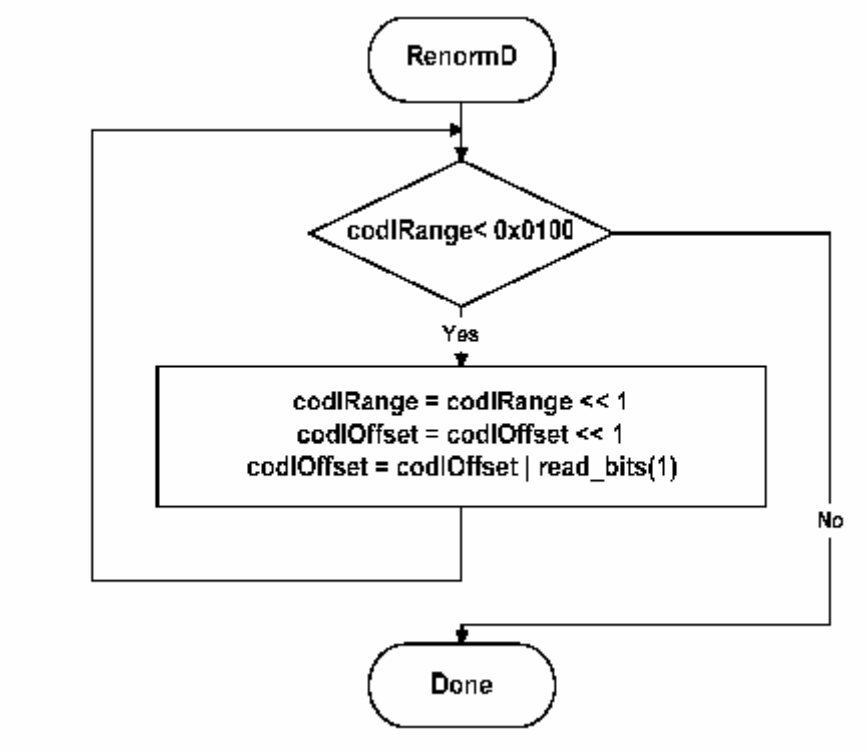
- For the control info in MB and Slice
  - ctxIdx = ctxIdxOffset+ctxIdxIncr
- For the residual data
  - ctIdx = ctxIdxOffset+ctxIdxBlockCatOffset+ctxIdxIncr

Binary Arithmetic Decoding

- Probability model
  - MPS
  - State
- Decoding Engine State Update
  - Range
  - Ioffset


DecodeBin(ctxIdx)
bypassFlag == 1?
Yes
No
DecodeBypass
ctxIdx==276?
Yes
No
DecodeTerminate
DecodeDecision(ctxIdx)
Done


DecodeDecision (ctxIdx)
qCodIRangeIdx = (codIRange>>6) & 3
codIRangeLPS = rangeTabLPS[pStateIdx][qCodIRangeIdx]
codIRange = codIRange - codIRangeLPS
codIOffset >= codIRange
Yes
No
binVal = !valMPS
codIOffset = codIOffset - codIRange
codIRange = codIRangeLPS
binVal = valMPS
pStateIdx = transIdxMPS[pStateIdx]
pStateIdx == 0?
Yes
valMPS = 1 - valMPS
No
pStateIdx = transIdxLPS[pStateIdx]
RenormD
Done


RenormD
codIRange< 0x0100
Yes
codIRange = codIRange << 1
codIOffset = codIOffset << 1
codIOffset = codIOffset | read_bits(1)
No
Done

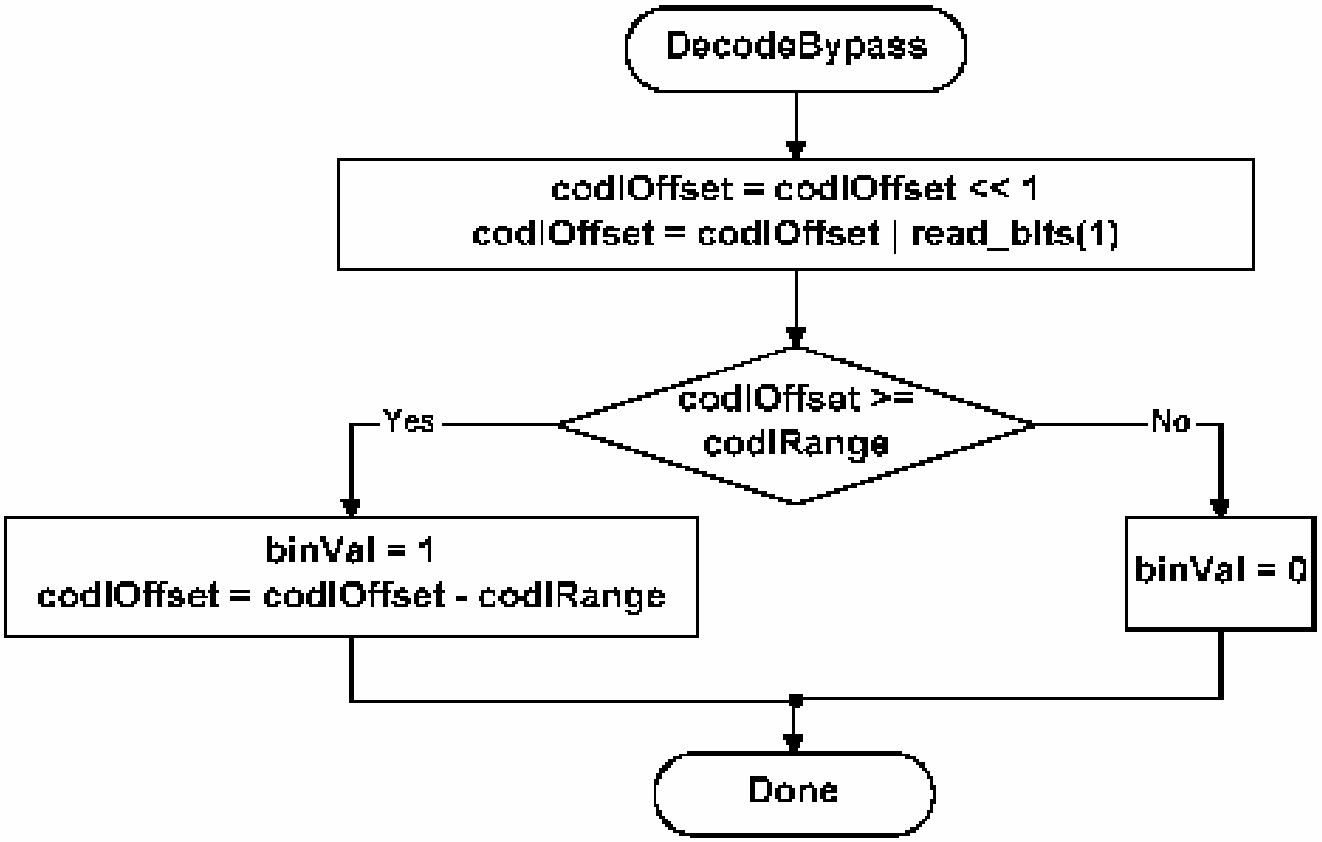


Figure 9-5 – Flowchart of bypass decoding process