AN UVLC ENCODER ARCHITECTURE FOR H.26L

Tu-Chih Wang, Hung-Chi Fang, Wei-Min Chao, Hong-Hui Chen and Liang-Gee Chen

DSP/IC Design Laboratory, EE department, National Taiwan University

ABSTRACT

Variable length code (VLC) is widely used in various compression applications. The latest under development video coding standard, H.26L, uses an unique pattern VLC code which is called UVLC in the test model document. In this paper, an UVLC encoder architecture is proposed to meet the requirement of TML8. This architecture is composed of a code splitter, a first 1 detector, a shifter, a length accumulator, and a code mux/output. Modified code number technique and auto self alignment technique are used to simplify the architecture. By using these techniques, the size of this architecture is much smaller comparing to traditional VLC encoder. It has been synthesized in 0.35μ cell library. The size of this architecture is 980 gates while critical path is less than 6ns.

1. INTRODUCTION

Every video coding algorithm has an entropy coding part to eliminate statistical redundancy. Traditional video coding standards, like MPEG, H.261[1] and H.263 series (H.263[2], H.263+, H.263++), use VLC as the entropy coding tool. Every symbol has its VLC code according to the probability of occurrence. Thus, the VLC encoder must contain all of the VLC tables to look up VLC code and its length. A typical entropy coding block of these standards is shown in Fig. 1. Symbols are fed into various VLC tables to look up the codeword and length. Then the codeword and length information goes into the VLC encoder to form the final output codeword. The VLC encoder combines input codeword and the codeword in buffer, then outputs word by word. The advantage of this architecture lies in the fact that every kind of symbol could find a proper VLC table to fit its probability.

H.26L tries another way to solve the entropy coding problem. It uses an unique VLC pattern which is called UVLC. Since the VLC pattern is fixed, the optimal symbol probability is pre-determined. And because the probability of each symbol's occurrence may not be the optimal distribution to this VLC pattern, H.26L combines more than one symbols to form a new symbol whose probability distribution is similar to this pattern. The entropy coding block for H.26L could be illustrated as Fig. 2. Symbols are processed by the probability transform tables to form a code number. The probability distribution of this code number is then nearly optimal to the UVLC pattern. UVLC encoder accepts code number and output codeword word by word. The advantage of UVLC is the simplicity of final output which facilitates encoder and decoder design. Moreover, UVLC could be reversely decoded because it is a kind of RVLC (Reversible VLC) pattern.

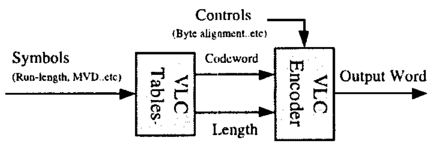

Fig. 1. Entropy coding block for JPEG, MPEG, H.261 and H.263 series video coding standards

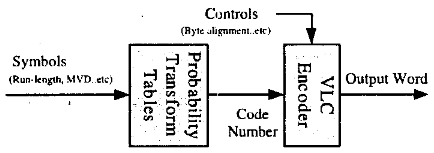

Fig. 2. Entropy coding block for H.26L

Previous state-of-the-art VLC encoder design could be found in [4], [6] and [7]. All of these VLC encoder architecture are parallel architecture. Bit-serial approach requires very high operating speed at the output stage and the throughput is much lower than parallel architecture. Lei[6] used three shifters to achieve faster operation. But the drawback is the larger gate count. Mario[7] used more compact VLC encoder architecture for JPEG encoder design. However, the complexity of this VLC encoder has been moved to the multiplexer control, which is not depicted in [7].

This papaer is organized as follows. New H.26L algorithms related to this design are described in section 2. Techniques used to simplify the architecture are illustrated in section 3. Section 4 shows the proposed UVLC architecture. Implementation result and the comparison with other VLC encoders are shown in section 5. Finally, a conclusion is given in section 6.

$$
\begin{gathered}
1 \\
0\,x_0\,1 \\
0\,x_1\,0\,x_0\,1 \\
0\,x_2\,0\,x_1\,0\,x_0\,1 \\
0\,x_3\,0\,x_2\,0\,x_1\,0\,x_0\,1 \\
\cdots\cdots\cdots\cdots\cdots\cdots
\end{gathered}
$$

Fig. 3. Rule of UVLC coding pattern

2. UVLC IN H.26L TEST MODEL

UVLC coding pattern is illustrated in Fig. 3, which could be found in [3]. It can be treated as two parts, the information part and the end bit. The information part is composed of two bits segments with leading bit 0. The second bit could be 0 or 1 which appears in Fig. 3 as x_n. The end bit is always a 1 to indicate a codeword's end. Codewords are sorted by length and information of $x_n \dots x_0$ to assign code numbers. The relation of code number and codeword is shown in Table 1.

Header contains information of temporal reference(TR, 8bit), picture QP(PQP, 5bit), format(1 bit) and end of sequence(EOS, 1bit). Total information is 15 bits long. The information is stuffed into $x_{14} \dots x_0$ to form a 31 bits codeword. Header coding doesn't follow code number rules and needs to be processed seperately.

Table 1. UVLC code table

Code Number	Codeword
0	1
1	0 0 1
2	0 1 1
3	0 0 0 0 1
4	0 0 0 1 1
5	0 1 0 0 1
6	0 1 0 1 1
7	0 0 0 0 0 0 1
8	0 0 0 0 0 1 1

3. MODIFIED CODING NUMBER TECHNIQUE

An UVLC encoder must extract codeword information and code length information from code number. But there is no easy way to extract codeword and code length from code number due to their complex relations. The circuit converting code number to codeword and code length will be large. Since code number is an intermediate signal looked up from the probability transform tables. We could change code number to reduce hardware implementation cost by adjusting code number in probability transform tables. And

Table 2. Modified code table

Code Number	Binary	Codeword
1	00001	1
2	$0001\underline{0}$	$0\ \underline{0}\ 1$
3	$0001\underline{1}$	$0\ \underline{1}\ 1$
4	$001\underline{00}$	$0\ \underline{0}\ 0\ \underline{0}\ 1$
5	$001\underline{01}$	$0\ \underline{0}\ 0\ \underline{1}\ 1$
6	$001\underline{10}$	$0\ \underline{1}\ 0\ \underline{0}\ 1$
7	$001\underline{11}$	$0\ \underline{1}\ 0\ \underline{1}\ 1$
8	$01\underline{000}$	$0\ \underline{0}\ 0\ \underline{0}\ 0\ \underline{0}\ 1$
9	$01\underline{001}$	$0\ \underline{0}\ 0\ \underline{0}\ 0\ \underline{1}\ 1$

there will be no overhead since the size of probability transform tables remains the same.

$$Number_{opt.} = Number_{org.} + 1 \tag{1}$$

The optimized code numbers are shown in Table 2. The relation between original code number and optimized code number is shown in equation (1). The binary expressions of optimized code numbers are also shown in Table 2. We could observe that the underline part of the binary expression column is equal to the underline part of codeword column. And the first 1's location of optimized code could reveal the codeword length. Equation (2) shows the relation between optimized code number and code length. For example, the binary expression of code number 7 is 00111. The location of first 1 is 2 bit left from LSB. The code length could be calculated as $2 \times 2 + 1 = 5$. From Fig. 3, the codeword pattern of length 5 is $0x_1 0x_0 1$. The left two bits of 7(11) are then stuffed into x_1 and x_0 to form the codeword 01011.

$$Length = First\ 1's\ Location \times 2 + 1 \tag{2}$$

4. PROPOSED UVLC ENCODER ARCHITECTURE

Fig. 4 shows the proposed UVLC encoder architecture. It composed of a first 1 detector to transform modified code number to length of the codeword, a code splitter inserting zero between information bits, a length accumulator to control the shifter to right location, a shifter to align codeword with output buffer, and a code mux/output block to combine codeword in buffer and input codeword. The function of each block is discussed as follows.

4.1. First 1 Detector

The function of this block is to get the code length from the location of the first 1. The truth table is shown in Table 3. Note that the output is always increased by two because the code length of UVLC is always an odd.

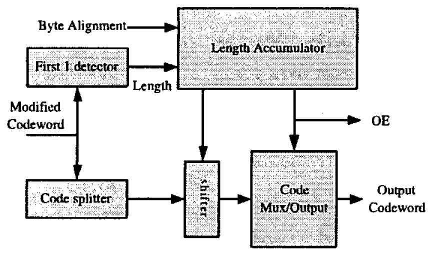

Fig. 4. Proposed UVLC encoder architecture

Table 3. Truth table of first 1 detector

Input	Output
0000 0000 0000 0001	1
0000 0000 0000 001x	3
0000 0000 0000 01xx	5
0000 0000 0000 1xxx	7
0000 0000 0001 xxxx	9
0000 0000 001x xxxx	11
0000 0000 01xx xxxx	13
0000 0000 1xxx xxxx	15
0000 0001 xxxx xxxx	17
0000 001x xxxx xxxx	19
0000 01xx xxxx xxxx	21
0000 1xxx xxxx xxxx	23
0001 xxxx xxxx xxxx	25
001x xxxx xxxx xxxx	27
01xx xxxx xxxx xxxx	29
1xxx xxxx xxxx xxxx	31

4.2. Code Splitter

Code splitter is a wiring box and does not contain any gate. Its function is simply adding 0 between modified code number and adding one bit 1 at LSB. MSB of the input is thrown because it doesn't contain any information. For example, if the input is 001 0000 1010 1111, the output will be 0001 00000000 01000100 01010101 1. The output of code splitter is the same as output codeword except the first 1. This additional 1 will be eliminated at the stage of code mux/output.

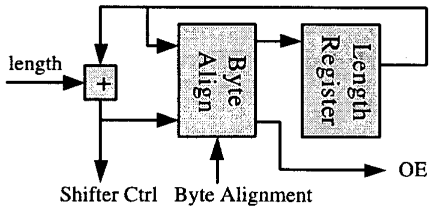

Fig. 5. Length Accumulator with byte alignment

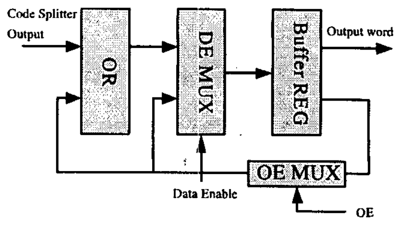

Fig. 6. Code mux/output

4.3. Length Accumulator

Fig. 5 is the architecture of length accumulator. Traditional length accumulator only contains an adder and a register. Carry out of the adder is treated as output enable signal. The length accumulator here is added some logic to complete byte alignment function. If byte alignment signal is high, the byte alignment block will get input data from length register and add it to the multiplies of 8. If byte alignment signal is low, the byte alignment block will get input data from adder and pass it to the register.

4.4. Shifter

The shifter is a barrel shifter. The width is the same as output buffer. In most of VLC architecture, barrel shifter is the largest part in the whole architecture. The size of this shifter is much related to total gate count of this architecture. The minimum value is 31 bits which is according to header length. But we set it to 48 bits in order to avoid overflow.

4.5. Code mux/output

Fig. 6 shows the architecture of code mux/output block. It is constructed by one OR plain, two multiplexer plains and one output buffer register. OR plain is used to combine codeword in output buffer and codeword from code splitter. Input codewords are stuffed from MSB to LSB in output buffer. If valid number of bits in output buffer is larger than 16, OE will be high to indicate output codeword (16 bits from MSB) is valid. And output buffer should be shift left by 16 bits. This job is done by OE MUX. DE MUX is used to form a feedback loop if input at this cycle is not valid. Since OE MUX is in the feedback loop of DE MUX, codeword output will be independent to valid data input. For example, if the input is a header at first cycle, a 19 bits codeword at second cycle, and no valid data at third cycle, the output will be 16 bits of header at first cycle, 15 bits of header and 1 bit of 19 bits codeword at second cycle, and 16 bits of 19 bits codeword at third cycle. Finally, the output buffer contains 2 bits of 19 bits codeword from LSB. From the discussion above, it could be known that the maximum

Output Buffer	001 1 01011
Code Splitter	1 011
OR Plain Out	001 1 01011 011

Fig. 7. OR plain example

throughput of this architecture is 16 bits per clock cycle.

Another function for the OR plain is to eliminate the first 1 output from code splitter. This is automatically done by the codeword alignment. Since the codeword is aligned to the output buffer, the first 1 of the code splitter output will align to last 1 of the output codeword buffer. The OR plain will eliminate this additional 1. An example is shown in Fig. 7. The output buffer contains codewords "001" "1" "01011". Now a new input codeword "011" with leading 1 is outputted from code splitter. The OR plain will combine these codeword to "001 1 01011 011". Note that the leading 1 in input codeword is combined to the tail 1 of "01011". This technique requires no additional circuit to deal with the first 1 problem from code splitter and is very efficient.

4.6. Header Consideration

The header of H.26L is a 31 bits UVLC pattern. But it doesn't use code number concept. The information of header is directly inserted into the information part of UVLC. In this architecture, MSB of input is reserved to indicate header. If MSB is set, the code length will be 31 according to Table 3. And other 15 bits could be filled up with header information. Since the bits after first 1 indicates information part of UVLC in our architecture, header output could be integrated in this architecture with no additional overhead.

5. IMPLEMENTATION AND COMPARISON

To reduce design period and provide technology independent portability, HDL design flow is used. This architecture has been synthesized in TSMC 0.35μ technology. The gate count is 980 while delay constrain is set to 6ns. Comparing to other VLC architecture, this architecture is much smaller and the speed is also sufficient for today's application. Detailed comparison table is given in Table 4.

Table 4. Comparison of several VLC encoder architecture

	Critical path delay(ns)	Area (gates)
Lei's[6]	5.56	10000
JAGUAR[7]	5.38	4793
Chang's[4]	5.72	2601
Proposed	5.92	980

6. CONCLUSIONS

In this paper, an efficient and cost effective UVLC encoder architecture for H.26L is proposed. The architecture could process one symbol per clock cycle. And the maximum throughput is 16 bits per clock. It has been synthesized to gate level in TSMC 0.35μ technology and can operate at 166Mhz. This proposed architecture is very compact and uses less hardware resource by modifying codeword number and codeword alignment technique. Due to its good area-timing property, this architecture is suitable for H.26L video codec design.

7. REFERENCES

[1] Draft ITU-T Recommendation H.261, Video codec for audiovisual services at p × 64 kbit/s, ITU-T, 1993.

[2] Draft ITU-T Recommendation H.263, Video coding for low bitrate communication, ITU-T, 1997.

[3] ITU-T, *H.26L TML8 Document from http://standard.pictel.com*, Sep., 2001.

[4] Hao-Chieh Chang, Liang-Gee Chen, Yung-Chi Chang and Sheng-Chieh Huang, "A VLSI Architecture Design of VLC Encoder for High Data Rate Video/Image Coding", Proceeding of ISCAS 2000, vol. 4, pp. 398-401, May, 2000.

[5] Shaw-Min Lei, M.T. Sun, K.Ramachandran and S. Palaniraj, "VLSI Implementation of an Entropy Coder and Decoder for Advanced TV Application", Proceeding of ISCAS 1990, pp. 3030-3033.

[6] Shaw-Min Lei and Min-Ting Sun, "An Entropy Coding System for Digital HDTV Applications", IEEE Transaction on Circuits and Systems for Video Technology, vol. 1, no. 1, pp. 147-154, Mar., 1991.

[7] Mario Kovac and N. Ranganathan, "JAGUAR: A Full Pipelined VLSI Architecture for JPEG Image Compression Standard", Proceeding of IEEE, vol. 83, no. 2, Feb., 1995.

[8] Yuji Fukuzawa, Kouichi Hasegawa, Hirokazu Hanaki, Eiji Iwata and Takao Yamazaki, "A Programmable VLC Core Architecture for Video Compression DSP", Proceeding of IEEE workshop on signal processing systems-Design and Implementation, pp. 469, Oct., 1997.