H.264/AVC용 CAVLC 디코더의 구현 연구

봉재훈, 김원삼, 손승일

한신대 정보통신학과

a study on an Implementation of CAVLC Decoder for H.264/AVC

Jaehoon-dong bong · one-sam kim, sung-il sun

hanshin University

E-mail : snrnwl@hs.ac.kr

요 약

지상파 DMB등에서 많이 사용하고 있는 기술은 H.264이다. 이 H.264는 적은 비트율에 비하여 고해상도의 영상을 만들어 낸다. 이런 손실압축을 하기 위해서 인트라와 인터등과 같은 전처리 과정과 DCT(Discrete Cosine Transform), 양자화 등등이 존재하지만 H.264에서 실제로 압축이 되는 부분은 엔트로피코딩이다. H.264에서는 Exp-Golomb과 CAVLC(Context-Adaptive Variable Length Coding), CABAC(Context-Adaptive Binary Arithmetic Coding) 세 가지를 지원하고 있다. 이중 CAVLC는 테이블을 기반으로한 압축기법을 사용한다. 테이블을 이용할 때는 코드워드의 길이와 값을 비교하는 방식을 사용하게 된다. 이는 수 많은 메모리 접속으로 인한 전력소모와 연산지연을 가져온다.

본 논문에서는 전송된 비트스트림에서 데이터를 찾을 때 코드워드의 길이와 값을 테이블에 비교해서 찾지 않고 테이블에 존재하는 규칙을 수식화 하여 찾을 수 있도록 하였다. 이는 최초 '1'이 나올때까지의 '0'의 개수와 그 이후 존재하는 코드의 값을 이용하여서 각 단계에 필요한 데이터를 추출해 낸다. 위와 같은 알고리즘을 이용하여 VHDL언어로 설계하였다.

키워드

CAVLC, 테이블, VHDL, H 264

Ⅰ. 서 론

효율적인 영상처리를 지원하는 H.264는 여러 가지 기법을 이용하여 영상을 최소한의 비트스트림의 변환 시킨다. 이러한 과정에서 실질적인 압축이 이루어지는 과정이 엔트로피 코딩이다. 이 엔트로피 코딩은 Exp-Golomb, CAVLC, CAVAC 등의 각 모드별로 효율적인 엔트로피 코딩을 적요하게 된다. 이중 CAVLC는 테이블을 기반으로 한다. 미리 자주 사용되는 값들을 테이블로 작은 코드워드를 할당해 놓은 것이다. 이 테이블에 맞추어서 인코딩 시킴으로써 효율적인 압축이 이루어진다. 하지만 이런 테이블은 디코딩시에도 필요하며 표준에서 제안된 알고리즘은 코드워드의 길이의 길이와 값을 테이블과 비교해서 찾는 방식이다. 하지만 원하는 값이 메모리 마지막에 위치할 경우 수 많은 필요없는 연산을 수행하여야 한다. 이는 많은 메모리 접속으로 인한 전력소모와 연산지연으로 인한 속력저하를 일으키게 된다. 인코딩과 다르게 디코딩은 제안된 환경에서 사용하게 된다. 이때 적은 전력량과 저사양의 프로세서에서 효율적으로 작동할 수 있어야한다. 그렇게 하기 위해서는 테이블에서 효율적으로 원하는 데이터를 찾아내거나 테이블에 존재하는 규칙을 이용한 알고리즘으로 테이블을 사용하지 않고서 데이터를 찾아낼 수 있으면 된다. 이런 방법을 이용함으로써 불필요한 메모리 접속을 없애고 적은 연산 횟수로 저전력 소모와 고속 연산이 가능해진다. 이에 본 논문은 효율적인 디코딩 기법을 연구하고 이를 VHDL언어로 설계하여 보았다[1-3].

Ⅱ. 본 론

2.1 CAVLC 개요

CAVLC는 지그재그 스캔된 4x4 변환 계수들의 오차 블록을 인코딩 하는 데 사용되는 방법으로 다음과 같은 특징을 가지고 있다. 예측, 변환 그리고 양자화 이후에 블록들은 일반적으로 대부분 '0'을 포함하기 때문에 CAVLC는 연이어 있는

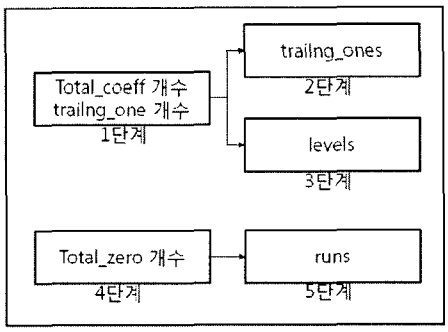
'0'을 간결하게 표현하기 위해 run-level 코딩을 사용한다. 그리고 지그재그 스캔 이후에 '0'이 아닌 가장 큰 계수들은 대개 ±1이고 CAVLC는 이를 'Trailing Ones'를 이용하여 인코딩한다.

또한 인접한 블록간의 상관관계가 있기 때문에 이를 이용하여 VLC look-up table을 선택하고 부호화한다. 마지막으로 '0'이 아닌 계수들의 레벨은 DC쪽에서 보다 큰 경향이 있다. 이러한 특징을 이용하여 이전에 인코딩된 레벨의 크기에 따라 VLC look-up 테이블을 선택하고 인코딩한다[3-5].

2.2 디코딩 순서

CAVLC는 크게 5단계로 이루어져 있으며,다른 디코딩 방식과 다르게 역순이 아닌 인코딩 된 순서 그대로 디코딩 하게 된다.

1단계에서 Total_coeff와 Trailing Ones의 개수를 구한 이후 2단계와 3단계가 수행된다. 즉 1단계에서 블록내에 존재하는 값의 개수만큼 다음 단계가 반복되는 것이다. 그리고 4단계에서 전체 '0'의 개수를 구하고 5단계에서 각 '0'이 아닌 계수 앞에 존재하는 '0'의 개수를 구한다.



(그림 1) 5단계 과정

2.3 기존 방식

표준에서 제시하고 있는 방식은 각 단계에서 필요한 코드워드를 추출해내기 위해서는 비트스트림을 일정부분 읽어들여서 그 길이와 값을 테이블에 비교하여 구하는 방식을 취하고 있다.

	0	1	2	3
0	1	-	-	-
1	0001 01	01	-	-
2	0000 0111	0001 00	001	-
3	0000 0011 1	0000 0110	0000 010	0001 1
4	0000 0001 11	0000 0011 0	0000 0101	0000 11
5	0000 0000 111	0000 0001 10	0000 0010 1	0000 100
6	0000 0000 0111 1	0000 0000 110	0000 0001 01	0000 0100
7	0000 0000 0101 1	0000 0000 0111 0	0000 0000 101	0000 0010 0
8	0000 0000 0100 0	0000 0000 0101 0	0000 0000 0110 1	0000 0001 00
9	0000 0000 0011 11	0000 0000 0011 10	0000 0000 0100 1	0000 0000 100
10	0000 0000 0010 11	0000 0000 0010 10	0000 0000 0011 01	0000 0000 0110 0
11	0000 0000 0001 111	0000 0000 0001 110	0000 0000 0010 01	0000 0000 0011 00
12	0000 0000 0001 011	0000 0000 0001 010	0000 0000 0001 101	0000 0000 0010 00
13	0000 0000 0000 1111	0000 0000 0000 001	0000 0000 0001 001	0000 0000 0001 100
14	0000 0000 0000 1011	0000 0000 0000 1110	0000 0000 0000 1101	0000 0000 0001 000
15	0000 0000 0000 0111	0000 0000 0000 1010	0000 0000 0000 1001	0000 0000 0000 1100
16	0000 0000 0000 0100	0000 0000 0000 0110	0000 0000 0000 0101	0000 0000 0000 1000

(그림 2) 첫 번째 테이블

(그림 2)에서 보는 것과 같이 코드 길이와 코드 값이 일치하는 곳에서 행과 열에 각각 존재하는 total_coeff값과 Trailing Ones의 값을 구하게 된다[1].

2.4 제안하는 방식

위에 존재하는 표를 최초 '1'이 나오기 전까지 존재하는 '0'의 개수와 그 뒤에 존재하는 코드로 (그림 3)~(그림 5)로 정리하였다.

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	0.0	1.1	2.2												13.1
10	2															
11	3				3.3	4.3										
100	4				2.1	5.3	6.3	7.3	8.3	9.3					16.0	
101	5				1.0		4.2	5.2	6.2	7.1						
110	6						3.1	4.1	5.1	6.1					16.1	
111	7						2.0	3.0	4.0	5.0					15.0	
1000	8										8.0	12.3	14.3	16.3		
1001	9										9.2	11.2	13.2	15.2		
1010	10										8.1	10.1	12.1	15.1		
1011	11										7.0	10.0	12.0	14.0		
1100	12										10.3	11.3	13.3	15.3		
1101	13										8.2	10.2	12.2	14.2		
1110	14										7.1	9.1	11.1	14.1		
1111	15										6.0	9.0	11.0	13.0		

(그림 3) 수정한 첫 번째 테이블

		0	1	2	3	4	5	6	7	8	9	10	11	12
1	1													15.3
10	2	1.1												
11	3	0.0												
100	4		4.3		7.3	8.3	5.0	9.3					16.3	
101	5		3.3		4.2	5.2	6.2	7.2					16.2	
110	6			5.3	4.1	5.1	6.1	7.1				14.2	16.1	
111	7			2.1	2.0	3.0	4.0	6.0				14.0	16.0	
1000	8			6.3					10.3	11.0	14.3	15.1		
1001	9			3.2					9.2	11.2	13.2	15.0		
1010	10			3.1					9.1	11.1	13.1	15.2		
1011	11			1.0					8.0	10.0	13.0	14.1		
1100	12									12.3	13.3			
1101	13								8.2	10.2	12.2	1		
1110	14								8.1	10.1	12.1			
1111	15								7.0	9.0	12.0			

(그림 4) 수정한 두 번째 테이블

		0	1	2	3	4	5	6	7	8	9
1	1										16.0
10	2									16.3	
11	3									16.2	
100	4								16.1		
101	5								15.0		
110	6								15.3		
111	7							13.1	15.2		
1000	8	7.3	5.1	3.0	7.0	12.3	12.0	15.1			
1001	9	6.3	5.2	7.2	6.0	11.2	13.2	14.0			
1010	10	5.3	4.1	7.1	9.2	10.1	12.1	14.3			
1011	11	4.3	4.2	2.0	5.0	9.0	11.0	13.0			
1100	12	3.3	3.1	9.3	10.3	11.3	13.3	14.1			
1101	13	2.2	8.3	6.2	8.2	10.2	12.2	14.2			
1110	14	1.1	3.2	6.1	8.1	9.1	11.1				
1111	15	0.0	2.1	1.0	4.0	8.0	10.0				

(그림 5) 수정한 세 번째 테이블

(그림 3) ~ (그림 5)와 같이 최초 '1'이 나오기 전까지 존재하는 '0'의 개수에 따라 뒤에 읽어야 할 코드길이를 알 수 있다. 이로써 한번에 코드를 읽어 드릴 수 있게 된다. 또한 정리된 표는 일정한 규칙을 가지고 있어서 이를 수식화 하면 테이블 없이도 원하는 값을 구할 수 있게 된다. 단, 4

번째 테이블은 6비트 고정길이 테이블로서 코드워드 '000011'의 경우에는 'total_coeff=0', 'trailing_one = 0' 이고 나머지 경우 앞의 4비트는 total_coeff를 나타내고 뒤의 2비트는 trailing_one를 나타낸다.

이로써 첫 번째 단계는 테이블을 사용하지 않고 수식을 이용한 알고리즘으로 모두 구현이 가능하다.

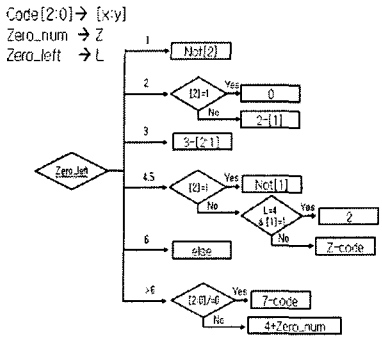
2.5 runs

run은 tota_zeros에 의해 zero_left값을 정하고 시작한다. run의 테이블은 zero_left의 값에 의해서 구분지어 진다. (그림 6)은 runs의 값을 테이블을 보여주고 있다..

run_before	zeros_left						
	1	2	3	4	5	6	>6
0	1	01	11	011	11	111	
1	0	01	10	010	10	000	110
2	-	00	01	001	011	001	101
3	-	-	00	000	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8	-	-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

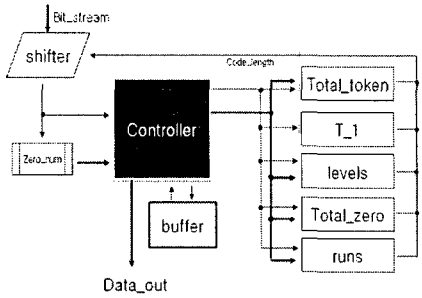
(그림 6) runs 테이블

(그림 6)과 같이 블록을 나누어서 각각의 경우로 처리할 경우 테이블 없이 고속의 연산이 가능해진다. (그림 6)는 (그림 7)을 순서도로 표현 한 것이다. runs에서도 zero_num의 개수와 그 이후에 코드를 이용하여서 원하는 값을 테이블 없이 구할수 있다. 각 단계는 zero_left에 의해서 나누어 지고 나오는 값에 의해서 zero_left의 값이 변하고 마지막 값을 처리 하지 않는다. total-zeros 과정을 거치는 이유이다[6-7].



(그림 7) runs 블록 순서도

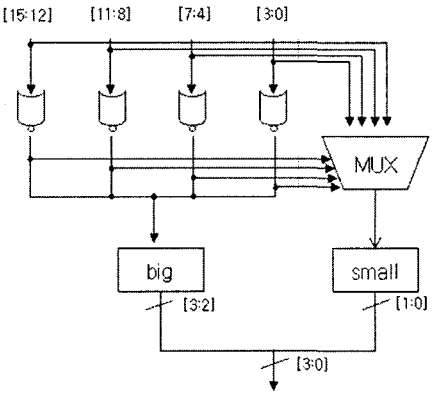
2.6 하드웨어적 구성



(그림 8) 전체 블럭도

비트스트림을 쉬프터를 이용하여 읽어 들여서 사용한 만큼 시프팅 시키고 코드워드를 읽어들이는 방식을 사용한다. 시프팅 된 값을에서 최초 '1'이 나오기 전의 '0'의 개수인 zero_num을 구한다. 쉬프팅 된 비트스트림과 zero_num 두개의 값을 이용해서 각 단계별로 디코딩 처리를 하고 데이터를 출력하게 된다.

2.7 zero_num



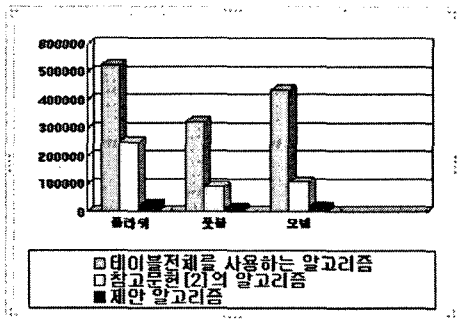
(그림 9) zero_num

zeor_num은 (그림 9)과 같이 16비트를 4비트씩 쪼개서 조합회로로 처리한 뒤 먹스를 통해서 '0'의 개수를 구한다.

V. 결 론

QCIF(176x144)크기의 플라워, 풋볼, 모빌 세 가지 실험 영상을 사용해서 얻어진 (그림7)의 결과를 실제 하드웨어에 적용 하였을 경우에도 동일한 효과를 볼 수 있음을 말한다. 제안하는 알고리즘 방식은 메모리 참조에 사용되는 모든 부분을

수식화 하여 메모리 참조를 0 %로 줄였다. 이로 인하여 메모리 참조로 인한 전력소모가 줄어들게 되었다. 또한 제안된 알고리즘은 모든 테이블을 사용하는 알고리즘에 비하여 유효 클럭 수 또한 10%이하로 줄어들게 되고,. 20배 이상의 처리 속도가 가능해진다. 이로써 기존의 알고리즘들보다 전력소모와 처리속도에서 좋은 결과를 얻게 되었다.



(그림7) 유효 클럭 수 비교

감사의 글

이 논문은 2006년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2006-521- D00339)

참고문헌

[1] Joint Video TEam(JVT) of ISO/IEC MPEG &ITU-T VCEG (ISO/IEC JTC1/SC29/WG11 AND ITU-T SG16 Q.6) 8th Meeting: Geneva, Switzerland, 23-27 May, 2003.

[2] 이은정 "An Efficient implementation of CAVLC in H264 standard" 부산대학교 대학원 일반대학원 2004

[3] 최웅일, 전병우 "H.264 엔트로피 인코딩 기법"방송공학회 제7권 제3호, pp. 54-64, 2002. 9.

[4] Saied, R. Chakrabrati, C. "Scheduling for minimizing the number of memory access in low powerapplications" VLSI Signal Processing, IX, 1996. [Workshop on], 30 Oct.-1 Nov. 1996 Pages: 169-178.

[5] 1ITU-TRec.H.264/ISO/iec 11496-10,"Advanced Video Coding", Final Committee Draft, Document JVT-E022, September 2002.

[6] Thomas Wiegand, Gray J. Sullivan, Gisle Bjontegaard, and Ajay Luthra, "Overview of the H.264/AVC Video Coding Standard" IEEE Trans.Circuits and ystems for video technology, vol.9,pp. 287-290, July. 2003.

[7] Iain E.G Richardson, "H.264 and MPEG-4", 홍릉출판사, 2004.