논문 2008-45CI-2-4

계층적 그룹 기반의 CAVLC 복호기

(A Hierarchical Group-Based CAVLC Decoder)

함 동 현*, 이 형 표*, 이 용 석**

(Donghyeon Ham, Hyoung Pyo Lee, and Yong Surk Lee)

요 약

동영상 압축 기술은 오랜 기간 동안 연구되었으며 H.264/AVC는 최근에 사용되고 있는 동영상 압축 표준 중 가장 효율적인 동영상 압축 표준으로 알려져 있다. H.264/AVC의 베이스 프로파일에서는 무손실 압축과정으로 기존의 VLC(Variable Length Coding) 방식 대신에 압축 효율을 향상시킨 CAVLC(Context-Adaptive Variable Length Coding)라는 압축 방식을 사용한다. CAVLC 복호기는 기존의 VLC 보다 많은 VLC 표가 필요하기 때문에 하드웨어로 구현하기에는 많은 면적을 요구하게 되며 소프트웨어로 구현 시에는 표 탐색에 의해서 성능이 저하된다. 본 논문에서는 이러한 CAVLC 복호기의 소프트웨어에서의 성능 저하를 막기 위해서 VLC 표를 계층적으로 집단화하여 코드만으로 주소를 정하고 정해진 VLC 표를 한번만 참조하여 성능을 향상시키는 방법을 제안한다. 제안된 알고리즘은 C 언어로 모델링하였으며 ARM ADS1.2에서 컴파일하고 ARM9TDMI 프로세서 시스템을 Armulator를 이용하여 시뮬레이션하였다. 실험 결과, H.264/AVC 표준 참조 프로그램인 JM(Joint Model) 10.2 보다 약 80%의 수행 시간 단축을 보였으며 최근 논문에서의 산술연산 알고리즘보다 15%의 성능 향상을 보였다 .

Abstract

Video compression schemes have been developed and used for many years. Currently, H.264/AVC is the most efficient video coding standard. The H.264/AVC baseline profile adopts CAVLC(Context-Adaptive Variable Length Coding) method as an entropy coding method. CAVLC gives better performance in compression ratios than conventional VLC(Variable Length Coding). However, because CAVLC decoder uses a lot of VLC tables, the CAVLC decoder requires a lot of area in terms of hardware. Conversely, since it must look up the VLC tables, it gives a worse performance in terms of software. In this paper, we propose a new hierarchical grouping method for the VLC tables. We can obtain an index of codes in the reconstructed VLC tables by simple arithmetic operations. In this method, the VLC tables are accessed just once in decoding a symbol. We modeled the proposed algorithm in C language, compiled under ARM ADS1.2 and simulated it with Armulator. Experimental results show that the proposed algorithm reduces execution time by about 80% and 15% compared with the H.264/AVC reference program JM(Joint Model) 10.2 and the arithmetic operation algorithm which is recently proposed, respectively.

Keywords: H.264/AVC, VLC, CAVLC, Entropy Coding

I. 서 론

동영상은 매우 많은 정보를 가지고 있기 때문에 동영상 정보를 압축과정 없이 저장하고 전송하는 것은 불가능하다. 압축 기술은 오랜 기간 동안 연구되었다. H.264/AVC[1]는 최근에 사용되고 있는 동영상 압축 표준 중 가장 효율적인 동영상 압축 표준으로 알려져 있다[2]. H.264/AVC는 많은 압축 기술을 적용하였다. 그중 VLC(Variable Length Coding) 방식은 좀 더 효율적인 CAVLC(Context-Adaptive Variable Length Coding) 방식과 CABAC(Context-Adaptive Binary Arithmetic Coding) 방식이 프로파일에 따라 선택적으로 이용된다[1]. CABAC은 CAVLC 보다 약 10% 정도의 더 높은 압축 효율을 보이나 계산 복잡도 때문에 휴대용 멀티미디어 기기 등에 적용되는 베이스라인 프로파일에 포함되

* 학생회원, ** 평생회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)

※ 본 연구는 한국과학기술재단 특정기초연구(No. R01-2006-000-10156-0)지원으로 수행되었으며, IDEC (IC Design Education Center)에 의해 지원되는 EDA 툴이 사용되었습니다.
접수일자: 2007년5월29일, 수정완료일: 2008년2월29일

지 않는다. CAVLC는 주변 상황에 적합하게 VLC 방법을 변형하여 적용한다. VLC 방법은 사용하는 VLC 표로 나타낼 수 있으며 따라서 CAVLC 에는 기존에 VLC 보다 많은 VLC 표를 사용하게 된다.

VLC 복호 방식은 크게 트리(tree) 기반의 방식과 비트 병렬 방식으로 나뉜다. 트리 기반 방식은 한 비트씩 읽어가며 뿌리(root)부터 잎(leaf)까지 찾아가는 방식으로 코드의 길이가 길어지면 복호 시간이 오래 걸린다. 반면에 비트 병렬 방식은 여러 비트를 동시에 읽어 참조표를 이용하여 한 번에 한 심벌(symbol)씩 복호하는 방식이다. 하지만 비트 병렬 방식의 경우 참조표의 크기에 의해 자원이 많이 필요한 단점이 있다.

이러한 방법들을 부분적으로 적용한 CAVLC 복호기 구현 연구들이 제안되었다. CAVLC 복호기는 하드웨어로 구현할 경우 빠른 복호 성능을 보여주는 반면 많은 VLC 표가 필요한 CAVLC 특성으로 인해 참조표의 크기가 커져서 많은 하드웨어 면적이 요구된다. 반면 소프트웨어로 구현할 경우 VLC 표 탐색에 많은 시간을 할당하게 되어 적합하지 못한 성능을 보여준다. VLC 표 탐색 방법에는 H.264/AVC의 표준 참조 프로그램인 JM[6]에서 사용한 순차적 검색 방식과 이진 찾기 방식이 있다. VLC 표 탐색 방법으로 $O(n)$의 복잡도를 갖는 순차적 검색 방식 대신 $O(\log_2 n)$의 복잡도를 갖는 이진 찾기 방식을 사용하여 성능을 향상시킬 수 있다. 하지만 이진 찾기 방식 또한 VLC 표를 여러 번 참조하기 때문에 높은 성능을 보여주지는 못한다. 이러한 문제를 해결하기 위해 [3]에서는 VLC 표를 분리하여 표 탐색 범위를 줄이는 방법을 사용하였다. 하지만 분리 조건의 규칙성이 떨어지고 분리된 VLC 표의 개수가 많아 조건 분기가 많은 단점이 있다. [4]에서는 확률이 높은 코드에 대해 코드와 심벌 간의 산술연산 관계를 찾아내서 표 탐색 시간을 부분적으로 줄였으며 [5]에서는 코드들을 여러 집단으로 나눠서 대부분의 심벌과 코드 간의 산술연산 관계를 찾아 VLC 표 참조의 횟수를 줄이고 성능을 향상시켰다. 하지만 코드와 심벌 간의 상관관계가 적은 CAVLC 특성으로 인해 복잡한 산술연산을 필요로 한다는 단점을 가진다.

본 논문에서는 CAVLC가 적용되는 분야를 고려하여 많은 VLC 표에 의해 하드웨어 자원이 많이 필요한 하드웨어 방식을 피해 기존의 산술연산 방식의 단점을 개선한 소프트웨어 방식을 제안한다. 상관관계가 적은 코드와 심벌 간의 관계를 보지 않고 코드와 주변 환경 조건만을 분석하여 코드를 분리하였다. 이를 바탕으로 VLC 표를 재구성하여 그 결과 주소를 정하는 과정이 간단해졌으며 VLC 표 참조 횟수를 한번으로 제한할 수 있다.

본 논문의 나머지 부분들은 다음과 같이 구성되었다. 본론에서는 CAVLC와 제안하는 알고리즘에 대해 설명하고 제안하는 알고리즘을 적용한 VLC 표를 보여준다. 실험에서는 JM 10.2[6]와 개선된 JM의 알고리즘[5], 산술연산 알고리즘[5], 제안하는 알고리즘을 시뮬레이션하고 결과를 비교한다. 마지막으로 결론에서는 실험결과에 대해 분석하고 제안하는 알고리즘을 평가한다.

Ⅱ. 본 론

1. CAVLC

CAVLC는 총 다섯 개의 값을 복호한다. 0이 아닌 계수의 개수(TotalCoeff), 마지막으로 연속해서 나오는 절대값이 1인 계수의 개수(TrailingOnes), 레벨(level), 런(run_before), 런의 총합(total_zeros) 을 주위의 상황에 적합한 VLC 표를 사용하여 복호한다. 복호된 값들을 이용하여 4x4 블락의 계수 정보를 정한다[1]. 레벨의 경우 코드와 심벌 간의 상관관계가 높아서 JM 표준 참조 소프트웨어에서도 VLC 표를 사용하지 않고 산술연산만으로 복호한다[6]. 따라서 본 논문에서는 레벨을 제외한 4개의 값들에 대한 복호 방법만 제안한다.

2. 제안하는 방식

CAVLC는 주변상황에 따라 다른 VLC 표를 사용하기 때문에 심벌과 코드간의 상관관계가 비교적 적다. 이로 인해 산술연산[5]을 이용한 처리 방식에서는 코드를 복호하기 위해서 복잡한 산술연산을 사용해야만 했다. 우리는 복잡한 연산으로 인해서 성능이 저하되는 것을 막기 위해 간단한 산술연산과 한 번의 VLC 표 참조만으로 코드를 복호하는 방식을 제안한다. 간단한 산술연산은 VLC 표 상에 참조할 주소를 생성하는데 사용한다. 생성된 주소를 이용하여 VLC 표를 참조하여 비트열 코드를 복호한다.

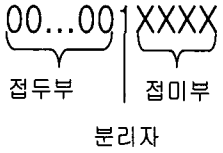

그림 1. 코드 구성
Fig. 1. Composition of the code.

주소를 생성하는 과정을 간단히 하고 VLC 표의 크기를 줄이기 위해 코드들을 분리하였다. 심벌과 코드간의 상관관계가 많은 부분은 따로 분리 하여 VLC 표를 이용하지 않고 산술연산만으로 코드를 복호하도록 하였다. 두개의 계층으로 나뉘어 코드를 분리하였다. 코드의 앞부분에 연속적으로 나오는 비트 '0'(접두부)의 길이와 처음 나오는 비트 '1'(분리자) 다음에 나오는 코드(접미부)의 길이를 코드 분리 요소로 사용하였다. 그림 1은 코드의 구성을 보여준다. 접미부의 길이에 의해 첫 번째 단계의 집단(대집단)이 분리된다. 접두부의 길이는 두 번째 단계의 집단(소집단)을 분리하는 데 사용하였다. 복호 과정에서 비트열의 접두부의 길이에 의해 대집단이 구분된다. 각각의 대집단은 같은 길이의 접미부를 가지는 소집단으로 구성되어 있어 대집단 내의 소집단의 주소를 간단한 산술연산만으로 계산할 수 있다.

VLC 표를 기본적으로 다음과 같은 순서로 분리한다. 첫째, 코드들을 접두부 길이의 오름차순으로 정렬 한다. 특별한 하드웨어 없이 비트열로부터 바로 알 수 있는 정보는 접두부의 길이 뿐이므로 이 과정을 통해 복호 과정에서의 코드 구분을 쉽게 할 수 있다. 둘째, 접두부의 길이가 같은 코드들을 모아 소집단을 분리한다. 셋째, 소집단을 구성하는 코드들의 접미부의 길이가 다를 경우 코드들의 접미부 길이를 소집단 최대 접미부 길이에 맞춘다. 코드의 접미부 길이를 n 만큼 증가시키고 싶을 때에는 다음과 같이 한다. 코드의 개수를 2^n 만큼 늘리고 각각의 코드 뒤에 0부터 2^n-1까지 n 비트의 코드를 붙여 코드를 분리한다. 따라서 소집단 내의 코드들은 모두 같은 길이의 접미부를 갖게 된다. 넷째, 소집단 내부의 코드들을 오름차순으로 정렬한다. 소집단 내부의 코드들은 모두 같은 접두부를 가지고 있으며 같은 길이의 접미부를 가지고 있기 때문에 접미부를 소집단 안에서의 주소로 쓸 수 있다. 다섯째, 소집단의 접미부의 길이가 변하는 부분을 경계로 하여 대집단을 분리한다. 대집단 안에서 소집단들의 접미부 길이는 모두 동일하다. 이는 소집단의 크기가 모두 동일하다는 것을 의미하며 소집단의 크기는 2의 접미부의 길이 제곱이다. 따라서 소집단의 오프셋 주소를 쉬프트 연산만으로 결정할 수 있다. 여섯째, 대집단이 적은 소집단만으로 이루어졌을 경우 인접한 대집단의 접미부 길이에 맞게 접미부의 길이를 늘여 인접한 대집단에 포함시킨다. 이를 통해 전체 대집단의 수를 줄인다. 전체 대집단의 수는 복호 과정에서의 분기와 이어지기 때문에 적을수록 성능에 좋은 영향을 준다. 마지막으로 코드와 심벌 간

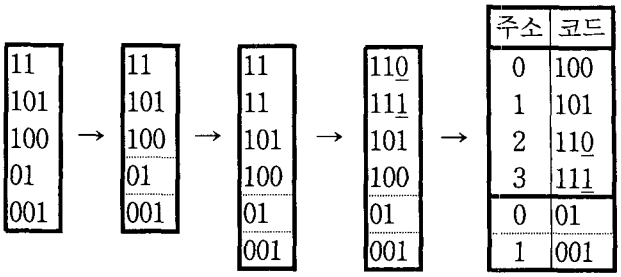

그림 2. VLC 표 재구성 과정 예
Fig. 2. Example of reconstructing VLC table.

의 상관관계가 높은 집단의 경우 별도의 VLC 표를 두지 않고 산술연산만으로 비트열을 복호할 수 있도록 산술연산 과정을 정의한다. 또한 코드를 통해서 쉽게 주소를 얻을 수 있는 집단은 특별히 주소를 계산하는 과정을 정의한다.

대집단은 각각 분리된 VLC 표를 이루며 생성된 주소는 코드의 정보가 담겨있는 VLC 표에서의 주소가 된다. VLC 표에는 코드의 복호 값과 접미부의 길이를 맞추기 전의 실제 코드의 접미부의 길이를 저장한다.

그림 2는 제안한 방법으로 VLC 표를 재구성하는 과정을 보여준다. 점선으로 코드들을 소집단 단위로 분리하였다. 코드의 굵은 폰트 부분은 실제 코드부분이며 밑줄 처리된 부분은 접미부 길이를 맞추기 위해서 추가된 코드이다. 굵은선으로 대집단을 분리하였다.

제안한 방식대로 재구성한 VLC 표를 이용하여 코드를 복호하는 과정은 다음과 같다.

첫째, 비트 1 이 나올 때까지 비트열을 읽어 접두부의 길이 L을 결정하고 현재 비트열의 위치를 갱신한다.

둘째, L을 각각의 대집단의 최소 접두부 길이 ($MinL_i$)와 비교해 코드가 속해있는 대집단 i를 결정한다. 대집단이 선택되면 참조할 VLC 표 $Table_i$가 결정되고 접미부의 길이 R_i가 정해진다.

셋째, R_i를 이용해 현재 비트열로부터 R_i의 길이를 갖는 접미부 R을 읽는다. 접미부는 소집단 내에서의 주소를 의미한다.

넷째, 코드는 대집단 i의 $L-MinL_i$ 번째 소집단에 속해있으며 대집단 i 안에 있는 소집단 들은 접미부의 길이 R_i에 의해서 2^{R_i}개의 코드들을 가진다. 따라서 코드가 속해있는 소집단의 오프셋 *offset*은 $(L-MinL_i)$ $<<R_i$ 를 계산하여 얻는다.

다섯째, *offset*+R 통해 선택된 VLC 표 $Table_i$ 내에서의 코드의 주소 $addr$를 계산한다.

여섯째, $Table_i[addr]$을 통해 코드를 복호하고 코드

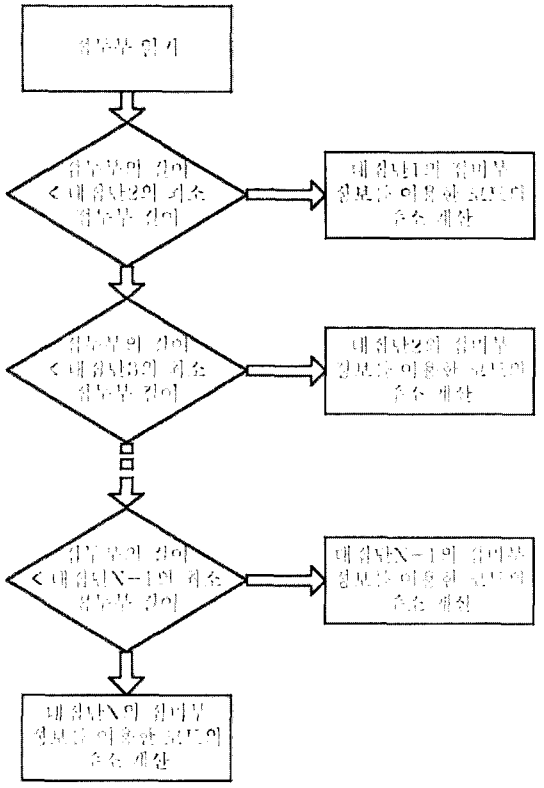

그림 3. 복호 과정 순서도
Fig. 3. Flow chart of the decoding process.

길이 정보를 이용해 비트열의 위치를 갱신한다.

그림 3은 앞에서 설명한 복호 과정을 순서도로 간단하게 나타낸 것이다. 대집단의 수가 적기 때문에 반복적인 조건비교는 큰 영향을 주지 않는다.

3. 제안하는 VLC 표

가. TotalCoeff 와 TrailingOnes

TotalCoeff 와 TrailingOnes를 복호하는 데는 총 4개의 VLC 표를 사용한다. VLC 표는 주변 블락의 TotalCoeff 값의 평균 nC에 의해 선택된다. 크로마 블락의 경우 nC 값이 -1을 갖는다.

표 1은 nC가 0이상이고 2미만일 때의 VLC 표를 제안한 알고리즘을 이용해 분리한 VLC 표이다. 코드들은 접두부의 길이의 오름차순으로 정렬하였다. T1s와 TC는 각각 TrailingOnes와 TotalCoeff를 의미한다. 접두부의 길이에 의해서 14개의 소집단으로 분리하였으며 점선으로 구분하였다. 접두부의 길이가 3인 소집단에서 코드들의 밑줄 처리 된 부분은 접미부의 길이를 맞추기 위해서 추가된 코드이다. 접미부의 길이에 의해서 4개의 대집단으로 분리하였으며 굵은 선으로 구분하였다. 접두부의 길이가 13인 소집단의 밑줄 친 코드들은 대집단의 수를 줄이기 위해 추가된 코드이다. 음영처리 된

표 1. TC와 T1s VLC 표(0≤nC<2)
Table 1. TC and T1s VLC Table (0≤nC<2).

[T1s,TC] ,길이	코드	접미부	접두부 길이	접미부 길이
[0, 0], 0	1		0	0
[1, 1], 0	01		1	
[2, 2], 0	001		2	
[1, 2], 2	000100	00	3	2
[0, 1], 2	000101	01		
[3, 3], 1	000110	10		
[3, 3], 1	000111	11		
…	…	…	…	
[3, 9], 2	00000000100	00	8	
[2, 7], 2	00000000101	01		
[1, 6], 2	00000000110	10		
[0, 5], 2	00000000111	11		
[0, 8], 3	0000000001000	000	9	3
[2, 9], 3	0000000001001	001		
[1, 8], 3	0000000001010	010		
[0, 7], 3	0000000001011	011		
[3, 10], 3	0000000001100	100		
[2, 8], 3	0000000001101	101		
[1, 7], 3	0000000001110	110		
[0, 6], 3	0000000001111	111		
…	…	…	…	
[0, 16], 2	00000000000001000	000	13	
[0, 16], 2	00000000000001001	001		
[2, 16], 2	00000000000001010	010		
[2, 16], 2	00000000000001011	011		
[1, 16], 2	00000000000001100	100		
[1, 16], 2	00000000000001101	101		
[0, 15], 2	00000000000001110	110		
[0, 15], 2	00000000000001111	111		
[1, 13], 0	000000000000001		14	0

표 2. T1s와 TC의 복호를 위한 대집단과 소집단의 수
Table 2. The number of small group and big group for decoding T1s and TC.

nC	대집단	소집단
0≤nC<2	4	15
2≤nC<4	5	13
4≤nC<8	2	10
nC=-1	4	8

대집단의 경우 T1s 값과 TC 값이 접두부의 길이와 같기 때문에 별도의 VLC 표를 사용하지 않고 복호할 수 있다. 표 2는 나머지의 TotalCoeff 와 TrailingOnes에 대한 VLC 표들의 대집단과 소집단의 개수를 나타낸다.

표 3. Total Zeros VLC 표
Table 3. Total Zeros VLC Table.

	TotalCoeff													
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	100	100	100	100	~~000~~	~~000~~	~~000~~	~~000~~	~~000~~	~~000~~	1	1	1	
011	101	101	101	101	001	001	001	001	001	001	01	01		
010	110	110	110	110	010	010	010	010	010	010	001			
0011	111	111	111	111	011	011	011	011	011	011				
0010	0100	0100	0100	0100	100	100	100	100	100	100	0001	001	01	1
00011	0101	0101	0101	0101	101	101	101	101	101	101	0000	000	00	0
00010	0110	0110	0110	0110	110	110	110	110	110	110				
000011	0111	0111	0111	0111	111	111	111	111	111	111				
000010	0010	0010	0010	0010	0001	0001	0001	0001	0001	0001				
0000011	0011	0011	0011	0011	00001	00001	00001	00001	00001	00001				
0000010	00010	00010	00010	00010	000001	000001	000001	000001	000001	000001				
00000011	00011	00011	00011	00011	000000	000000	000000	000000	000000	000000				
00000010	000000	000000	000000	000000										
000000011	000001	000001	000001	000001										
000000010	000010	000010	000010	000010										
000000001	000011	000011	000011	000011										

나. Total Zeros

표 3은 Total Zeros에 대한 VLC 표를 제안한 방법에 기초하여 나눈 코드 표이다. Total Zeros는 TotalCoeff 값에 의해서 다른 VLC 표를 쓴다. 대집단 안에서 음영 처리된 코드들은 접미부 대신에 소집단 안에서의 주소 값으로 쓰인다. 취소선이 그어진 코드의 경우 주소 계산을 간단하게 하기 위해 의미 없이 들어간 코드이다. 심벌 값은 표에 표시하지 않았으며 실제 표에는 심벌 값과 코드의 길이가 저장된다.

다. Run

표 4는 Run에 대한 VLC 표를 제안한 방법에 기초해서 나눈 코드 표이다. Run은 Total Zeros 값으로 시작해서 Run 값으로 계속 뺀 값, zeroLeft에 따라 다른 VLC 표를 사용한다.

표 4. Run의 VLC 표
Table 4. VLC table for Run.

	zeroLeft					
1	2	3	4	5	6	>6
000	000	000	000	000	000	~~000~~
001	001	001	001	001	001	001
010	010	010	010	010	010	010
011	011	011	011	011	011	011
100	100	100	100	100	100	100
101	101	101	101	101	101	101
110	110	110	110	110	110	110
111	111	111	111	111	111	111
						0001
						00001
						000001
						0000001
						00000001
						000000001
						0000000001
						00000000001

III. 실 험

제안한 방법을 JM[6] 원본과 개선된 JM[5], 산술연산[5]의 방법과 비교하였다. 개선된 JM은 산술연산에서 비교 대상으로 쓰인 JM으로써 불필요한 비트열 참조 횟수를 줄였다. 실험은 ARM Armulator에서 arm9tdmi를 에뮬레이트 하여 이루어졌다. 표 5는 성능 비교하는데 사용한 동영상들을 보여준다[7]. 표준 참조 소프트웨어인 JM을 이용하여 비교에 사용된 동영상들을 복호하고 결과물을 비교해 검증하였다. 표 6은 시간 측정의 결과이다. 표준1과 표준2, 산술, 제안은 각각 JM 원본과 개선된 JM, 기존에 제안되었던 산술연산 방법, 우리가 제안하는 방법을 의미한다. 그림 4는 표 6을 그래프로 나타낸 것이다. 그래프는 제안하는 방식이 JM 보다 수

표 5. 입력 동영상
Table 5. Input sequence.

이름	크기	양자화 계수	생략 프레임	프레임 수
Container	QCIF	20	2	100
News	QCIF	24	2	100
Foreman	QCIF	24	2	100
Silent Voice	QCIF	16	1	150
Paris	CIF	20	1	150
Mobile	CIF	20	0	300
Tempete	CIF	28	0	300

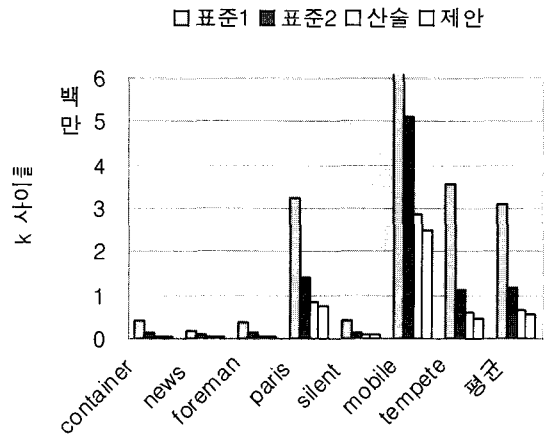

그림 4. 복호화 시간
Fig. 4. Decoding time.

표 6. 동영상 복호화에 소모된 시간 (k cycle)
Table 6. Execution time of the decoding process. (k cycle)

이름	표준1	표준2	산술	제안
container	400,185	121,919	65,454	51,801
news	200,418	73,519	41,797	35,183
foreman	366,905	120,864	66,677	53,943
paris	3,254,005	1,391,267	830,464	736,900
silent	439,766	155,502	87,228	73,928
mobile	13,385,818	5,119,518	2,835,972	2,470,869
tempete	3,546,604	1,105,716	591,191	472,648

행시간의 약 80 % 정도를 단축시켰으며 산술연산 방법보다도 약 15 % 정도의 시간을 단축한다는 것을 보여준다.

IV. 결 론

본 논문에서는 CAVLC 의 성능을 향상시키기 위해 CAVLC에서 사용하는 각각의 VLC 표를 코드만으로 쉽게 주소를 찾을 수 있도록 분리하고 개선하였다. 그 결과 코드의 간단한 산술연산만으로 VLC 표의 주소를 결정하고 VLC 표를 한번만 참조하여 코드를 복호할 수 있게 되었다. 이는 VLC 표 참조를 심벌 당 한번으로 제한하고 산술연산[5]에 비해 15% 성능을 향상시켰다. 또한 제안한 방식은 복호 과정에 있어서 산술연산 방식보다 규칙성을 가지고 있어서 응용프로그램에 특화된 명령어를 가지는 프로세서를 이용하여 좀 더 많은 향상을 가져올 수 있는 가능성을 가지고 있다.

참 고 문 헌

[1] JVT, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)," May 2003.

[2] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. on CWSVT*, Vol.13, No.7, July 2003.

[3] X. Quan, L. Jilin, W. Shijie and Z. Jiandong, "H.264/AVC Baseline Profile Decoder Optimization on Independent Platform," *Wireless Comm., Networking and Mobile Computing, 2005. Proc.. 2005 Int. Conf. on*, Vol.2, Iss., 23-26 Sept. 2005.

[4] Y. H. Moon, G. Y. Kim, and J. H. Kim, "An Efficient Decoding of CAVLC in H.264/AVC Video Coding Standard," *IEEE Trans. on Consumer Electronics*, Vol.51, No3, August 2005.

[5] Y.-H. Kim, Y.-J. Yoo, J. Shin, B. Choi and J. Paik, "Memory-Efficient H.264/AVC CAVLC for Fast Decoding," *IEEE Trans. on Consumer Electronics*, Vol.52, Iss.3, Aug. 2006.

[6] Karsten Suhring, H.264/AVC software JM10.2, *http://iphome.hhi.de/suehring/*

[7] G. Sullivan and G. Bjontegaard, "Recommended simulation common conditions for H.26L coding efficiency experiments on low-resolution progressive-scan source material," *ITU-T VCEG*, Doc. VCEG-N81, September, 2001.

저 자 소 개

함 동 현(학생회원)
2005년 성균관대학교 정보통신 공학부 학사 졸업.
2006년~현재 연세대학교 전기 전자공학과 석사과정.
<주관심분야 : 영상신호처리, SoC 설계>

이 용 석(평생회원)
1973년 연세대학교 전자공학과 학사 졸업.
1977년 University of Michigan Electrical Engineering 석사 졸업.
1981년 University of Michigan Electrical Engineering 박사 졸업.
1993년~현재 연세대학교 전기전자공학과 교수.
<주관심분야 : 마이크로프로세서 설계, VLSI 설계, DSP 프로세서 설계, 고성능 연산기 설계>

이 형 표(학생회원)
2001년 건국대학교 전자공학과 학사 졸업.
2001년~현재 삼성전자 DM총괄 선임연구원
2006년~현재 연세대학교 전기 전자공학과 석사과정.
<주관심분야 : 영상신호처리, SoC 설계>