논문 2009-46SP-5-18

멀티미디어 DSP를 위한 AVS 비디오 복호화기 구현

(AVS Video Decoder Implementation for Multimedia DSP)

강 대 범*, 심 동 규**

(Dae-Beom Kang and Dong-Gyu Sim)

요 약

Audio Video Standard (AVS)는 중국내의 멀티미디어 응용기기를 위해 개발된 오디오/비디오 압축 표준이다. AVS는 표준화 코덱 중 성능이 가장 우수한 것으로 알려진 H.264/AVC에 비해 낮은 복잡도의 비디오 알고리즘을 사용하면서도 비슷한 RD 성능을 보인다. AVS 비디오 코덱은 VGA급 이상의 영상을 타겟으로 하기 때문에 큰 해상도에서 압축효율이 좋은 8×8 단위 블록의 예측 및 변환 알고리즘을 사용한다. 현재 중국에서 IPTV 및 모바일 애플리케이션을 위한 코덱으로 AVS를 사용하는 비중이 높아지고 있어 국내의 기업 및 연구소에서도 AVS를 위한 애플리케이션 및 칩 개발을 위한 연구가 진행되고 있다. 본 연구에서는 AVS 비디오 복호화기 알고리즘을 분석하고 이를 바탕으로 하여 불필요한 메모리 연산이 없도록 AVS 비디오 복호화기를 구현하고 이를 TI의 Davinci EVM보드에서 최적화하였다. 또한, 제안한 복호화기에 고속의 VLD 알고리즘을 적용하고 linear assembly로 디블록킹 필터를 구현하는 등 DSP에 적합하도록 최적화를 진행하였다. 이를 통해 AVS의 참조 소프트웨어인 RM 5.2J 복호화기와 비교하여 500%~700%의 복호 속도 향상을 이루었다.

Abstract

Audio Video Standard (AVS) is the audio and video compression standard that was developed for domestic video applications in China. AVS employs low complexity tools to minimize degradation of RD performance of the state-the-art video codec, H.264/AVC. The AVS video codec consists of 8×8 block prediction and the same size transform to improve compression efficiency for VGA and higher resolution sequences. Currently, the AVS has been adopted more and more for IPTV services and mobile applications in China. So, many consumer electronics companies and multimedia-related laboratories have been developing applications and chips for the AVS. In this paper, we implemented the AVS video decoder and optimize it on TI's Davinci EVM DSP board. For improving the decoding speed and clocks, we removed unnecessary memory operations and we also used high-speed VLD algorithm, linear assembly, intrinsic functions and so forth. Test results show that decoding speed of the optimized decoder is 5~7 times faster than that of the reference software (RM 5.2J).

Keywords : AVS, DSP, VLD, DM6446, Davinci

I. AVS 개요

중국의 멀티미디어 시장은 2000년대 들어 급격하게 커지게 되었고, 이에 따라 국제 표준 기술의 사용에 대한 로열티 지출이 급증하게 되었다. 이를 해결하기 위해 2002년부터 셋톱 박스, PMP와 IPTV 등에서 사용하기 위해 중국내 자체 표준화 작업을 수행한 결과 Audio Video Standard (AVS)가 개발되었다. AVS는 로열티의 부담을 줄이기 위해 기존의 비디오 코덱에서 사용한 아이디어들을 일부 배제하여 H.264/AVC 대비 1/2의 복잡도를 가지면서도 VGA급 이상의 영상에서는 H.264/AVC의 베이스라인 프로파일와 유사한 압축 효율을 보이는 것으로 알려져 있다.

KISTI (http://www.cww.net.cn/)에서 조사한 바에 따르면 이렇게 개발된 AVS는 중국내에서의 IPTV에서 사용되고 있으며, 베이징, 상하이, 광저우 등의 도시들에 있는 휴대폰 TV업체들에서 표준적용을 위해 시험되

* 학생회원, ** 정회원, 광운대학교 컴퓨터공학과
(Dept. of Computer Engineering, Kwangwoon University)

※ 본 연구는 "서울시 산학연 협력사업"을 통하여 이루어졌음

접수일자: 2009년6월1일, 수정완료일: 2009년9월1일

고 있다[1]. 또한, P2P, 양방향 TV 및 휴대폰 TV와 관련된 사업을 하는 베이징의 광스퉁다 네트워크유한공사에서는 산하의 네트워크 플랫폼에 AVS 표준을 채택하는 등 갈수록 사용에 가속도가 붙을 전망이다[2].

AVS가 QCIF (176×144)급 및 CIF (352×244)급보다 큰 VGA (640×480)급 및 D1 (720×480)급 이상을 타겟 영상으로 삼고 있어, H.264/AVC 대비 낮은 복잡도를 가짐에도 불구하고 큰 영상에서의 실시간 복호에 어려운 점이 많다. 특히, 참조 소프트웨어인 RM의 경우 H.264/AVC의 참조 소프트웨어인 JM과 유사한 구조로 구현이 되어 DSP에서 포팅 시 수행 속도가 매우 느리다.

본 연구에서는 AVS 비디오 복호화기의 알고리즘을 이해하고 이를 바탕으로 구조적으로 효율적인 복호화기 소프트웨어를 설계하고 이를 C64x+ 계열의 Davinci (DM6446)에 최적화하였다. 또한, DM6446에서의 프로파일링 결과를 바탕으로 연산량을 분석하여 TI DSP 컴파일러에서 제공하는 intricsic을 사용하고 VLD에 대해 고속화 알고리즘을 적용하였다. 연산량이 큰 디블록킹 필터에 대해서는 기존의 DSP를 위한 최적화 알고리즘을 linear asm으로 적용하여 구현하고 이외에 산술 연산을 이용한 C언어 최적화를 수행함으로써 RM 5.2j와 비교하여 크게 향상된 복호 성능을 보인다.

Ⅱ장에서는 AVS 비디오 복호화기의 주요 알고리즘에 대해 설명하고, Ⅲ장에서는 제안한 AVS 비디오 복호화기에서 사용한 최적화에 대해 기술한다. Ⅳ장에서는 제안한 AVS 비디오 복호화기의 VLD 복호 성능을 RM 5.2j의 VLD 복호 성능과 비교해보고 최적화를 통한 전체적인 성능의 비교에 대한 결과를 요약하도록 한다.

Ⅱ. AVS 비디오 복호화기의 주요 특징

AVS 비디오 코덱의 주요 기술들은 기존에 많이 사용되어진 MPEG-2 및 H.264/AVC 까지의 비디오 코덱과 유사하다. 단, 세부 도구의 선택에 있어 기존의 기술을 피하고 기존 비디오 코덱과 대비하여 비트율의 변화 없이 PSNR은 높이면서 복잡도가 낮은 알고리즘들을 채택하였다.

그림 1은 AVS 비디오 복호화기의 전체 시스템 블록도이다. 시스템은 입력으로 들어온 비트스트림에 대해 엔트로피 복호화 후 역양자화 및 역 변환을 통해 공간

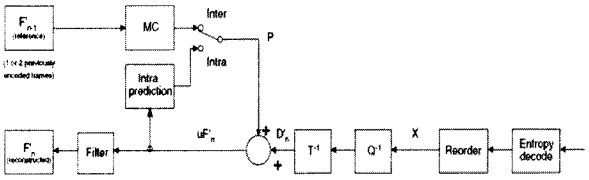


그림 1. AVS 비디오 디코더의 블록도
Fig. 1. Block diagram of the AVS video decoder.

축으로 데이터를 변환시키고, 화면 내/간 예측을 통해 복원된 영상을 만들어 낸다. 그림 1의 각 기능 블록 내부의 알고리즘은 8×8 블록 단위를 기반으로 하여 이루어진다.

1. 엔트로피 부/복호화 방법

AVS의 엔트로피 부호화는 텍스쳐 정보를 포함한 모든 문법 요소들에 k 차수 지수-골룸 부호화 방법을 사용한다. 표 1에서 보듯이 k 차수 지수-골룸 부호화 방법은 작은 값에 대해서는 짧은 비트를 사용하고 값이 커질수록 긴 비트를 할당함으로써 코딩 효율을 높이는 방법이다.

AVS에서는 이러한 k 차수 지수-골룸 부호화 방법을 사용함으로써 빠른 엔트로피 부호화 및 복호화가 가능하다. 차수는 0 차부터 3 차까지의 차수를 사용하여 빠른 엔트로피 부/복호를 하면서도 부호화 성능이 향상될 수 있도록 하였다.

또한, CBP와 텍스쳐 정보와 같은 문법 요소들은 작

표 1. k 차수 지수-골룸 부호
Table 1. k-th order Exp-Golomb codes.

차수	코드 구조	복호 범위
k = 0	1	0
	0 1 x_0	1-2
	0 0 1 x_1 x_0	3-6
	.....	.....
k = 1	1 x_0	0-1
	0 1 x_1 x_0	2-5
	0 0 1 x_2 x_1 x_0	6-13
	.....	.....
k = 2	1 x_1 x_0	0-3
	0 1 x_2 x_1 x_0	4-11
	0 0 1 x_3 x_2 x_1 x_0	12-27
	.....	.....
k = 3	1 x_2 x_1 x_0	0-7
	0 1 x_3 x_2 x_1 x_0	8-15
	0 0 1 x_4 x_3 x_2 x_1 x_0	16-47
	.....	.....

```
leadingZeroBits = -1;

for(b = 0; !b; leadingZeroBits++)
    b = read_bits(1);

CodeNum = 2^(leadingZeroBits + k) - 2^k + read_bits(leadingZeroBits + k);
```

그림 2. 지수-골룸 복호 방법
Fig. 2. Decoding method of Exp-Golomb.

표 2. P 프레임에서 휘도 블록의 run, level을 위한 VLD3_Inter 테이블
Table 2. VLD3_Inter table to decode level and run values of luma block.

Run	EOB	Level > 0									RefAbsLevel
		1	2	3	4	5	6	7	8	9	
	2	-	-	-	-	-	-	-	-	-	
0	-	0	3	7	13	17	27	35	43	55	10
1	-	5	11	21	33	51	-	-	-	-	6
2	-	9	23	37	57	-	-	-	-	-	5
3	-	15	29	47	-	-	-	-	-	-	4
4	-	19	41	-	-	-	-	-	-	-	3
5	-	25	49	-	-	-	-	-	-	-	3
6	-	31	-	-	-	-	-	-	-	-	2
7	-	39	-	-	-	-	-	-	-	-	2
8	-	45	-	-	-	-	-	-	-	-	2
9	-	53	-	-	-	-	-	-	-	-	2

은 값이 항상 작은 비트에 할당되지 않으므로 확률에 의거하여 맵핑 테이블을 사용하였다. 텍스쳐 정보의 경우에는 연속된 0의 값에 대한 정보인 run과 값의 크기 level을 묶어 (run, level) 형태의 2D방법을 이용하고 휘도, 색상, I 프레임 그리고 P 프레임에 대해서 테이블을 적응적으로 교체해 가며 사용함으로써 코딩 효율을 높였다. 또한, level의 크기가 59보다 크거나 같은 경우에는 테이블을 사용하지 않고 ESCAPE 모드를 두어 크기의 차이에 대해 부호화함으로써 가변 길이 부호 테이블이 불필요하게 커지는 것을 방지하였다.

부호화된 비트의 복호는 표 1을 테이블로 구성하여 복호할 수 있지만, 일반적으로는 그림 2와 같은 방법과 같이 연산으로 대체하여 복호를 한다.

여기서, leadingZeroBits는 비트스트림에서 연속한 0의 개수를 의미하며 k 차수를 고려하여 CodeNum을 생성한다. 생성된 CodeNum은 직접 값으로 사용되거나 가변 길이 부호 테이블과 동일한 가변 길이 복호 테이블에 맵핑하여 실제의 문법 요소로 사용된다. AVS는 화면 내 예측과 화면 간 예측, 그리고 각각의 휘도 및 색차 성분에 대한 오차 값을 복호하기 위해 13개의 테이블을 사용한다. 그 중에서 표 2는 화면 간 예측의 차분 신호를 복호하기 위해 사용되는 가변 길이 복호 테이블을 나타낸다. 예를 들어, CodeNum이 "57"로 복호되면 이 때의 Run 값은 2, Level 값은 4 그리고 RefAbsLevel 값은 5로 복호 된다.

2. 8×8 블록 단위의 정수 역변환

4×4 정수 변환의 경우 작은 영상에서는 변환에 의한 에너지 집중도가 높지만 해상도가 VGA (640×480)급 이상의 영상에서는 8×8 정수 변환의 에너지 집중도가 높다고 알려져 있다. AVS는 VGA급 이상의 영상에서 좋은 압축 성능을 얻는 것을 목표로 개발되었으며, 따라서 8×8을 최소 예측 블록 단위 및 변환 단위로 사용한다. 연산 복잡도를 낮추기 위해 MPEG-2/4와 달리 그림 3으로 근사화된 정수 변환 및 역변환을 구현하였으며, 16 비트 내에서는 오차 없이 완벽한 복원이 되도록 하였다. 또한, 복호화기에서의 복잡도를 낮추고자 스케일링을 비대칭으로 구성하여 부호화기에서만 스케일링이 존재하도록 하였다.

$$T_8 = \begin{bmatrix} 8 & 10 & 10 & 9 & 8 & 6 & 4 & 2 \\ 8 & 9 & 4 & -2 & -8 & -10 & -10 & -6 \\ 8 & 6 & -4 & -10 & -8 & 2 & 10 & 9 \\ 8 & 2 & -10 & -6 & 8 & 9 & -4 & -10 \\ 8 & -2 & -10 & 6 & 8 & -9 & -4 & 10 \\ 8 & -6 & -4 & 10 & -8 & -2 & 10 & -9 \\ 8 & -9 & 4 & 2 & -8 & 10 & -10 & 6 \\ 8 & -10 & 10 & -9 & 8 & -6 & 4 & -2 \end{bmatrix}$$

그림 3. 8×8 정수 역변환의 계수 행렬
Fig. 3. Coefficient matrix of 8×8 inverse integer transform.

3. 8x8 블록 단위의 화면 내 부호화

AVS는 8×8을 기본 예측 블록으로 화면 내 예측 및 화면 간 예측이 이루어지며 화면 내 예측의 경우 휘도 성분에 대해 그림 4와 같은 5가지 모드가 존재한다. H.264/AVC의 휘도 성분에 대한 모드인 4×4 모드 16개, 16×16 모드 4개와 비교해 볼 때 적은 수의 예측 방법을 사용함으로써 부호화기와 복호화기의 복잡도를 줄이고 예측 모드를 표현하기 위한 비트가 줄어들었다. 또한, 2번 모드인 DC의 경우 가중 평균을 이용함으로써

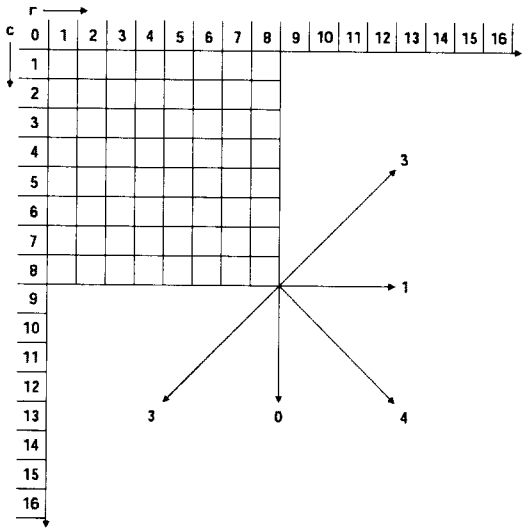


그림 4. 휘도 블록에 대한 화면 내 예측모드
Fig. 4. Intra prediction modes for Luma block.

H.264/AVC의 DC 모드와 차별화를 두었다.

4. 1/4 단위의 4-tap 보간법

최근의 비디오 코덱은 화면 간 예측을 향상시키기 위해 참조 영상을 1/2뿐만 아니라 1/4화소 단위까지 확장하여 예측의 정밀도를 높인다. AVS 비디오 코덱에서는 복잡도가 낮은 4-tap 계수를 사용하여 이를 구현하였다. 그림 5에서 볼 때 A, B, C 등은 정수 화소의 위치를 의미하며 a, b, c 등은 1/2 및 1/4 위치의 화소를 의미한다. 1/2 에 해당하는 화소의 경우 [-1 5 5 -1] 계수를 사용하며, 1/4 에 해당하는 화소는 [1 7 7 1] 계수를 사용하여 저주파 통과 필터링을 하게 된다. 색차 신호에 대해서는 bi-linear 필터를 이용하여 보간을 하

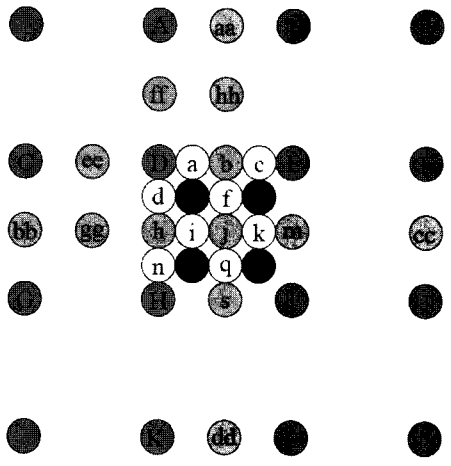


그림 5. 정수, 1/2 및 1/4 화소에 대한 위치
Fig. 5. Pixel positions (integer, half and quarter pels).

표 3. AVS의 보간 필터 식
Table 3. Interpolation filter formula of the AVS.

화 소	식
b	Clip{(-C+5D+5E-F+4) >> 3}
h	Clip{(-A+5D+5H-K+4) >> 3}
j	Clip{(-bb'+5h'+5m'-cc' + 32) >> 6} or Clip{(-aa'+5b'+5s'-dd' + 32)) >> 6}
a	Clip{(ee'+7D'+7b'+E' + 64) >> 7}
d	Clip{(ff'+7D'+7h'+H' + 64) >> 7}
i	Clip{(gg'+7h''+7j'+m'' + 512) >> 10}
f	Clip{(hh'+7b''+7j'+s'' + 512) >> 10}
e	Clip{(D''+j'+64) >> 7}
g	Clip{(E''+j'+64) >> 7}
p	Clip{(H''+j'+64) >> 7}
r	Clip{(I''+j'+64) >> 7}

게 된다.

표 3은 그림 5에 대해 화소 위치에 따라 수식으로 정리한 것이다. 표 3의 문자 뒤에 " ′ "는 8배 스케일링 된 값을 의미하며 " ″ "는 64배 스케일링 된 값을 의미한다.

5. 디블록킹 필터

디블록킹 필터는 블록 단위의 변환 및 양자화 때문에 생기는 블록 경계의 블록킹 현상을 제거하기 위해 사용한다. AVS에서 사용하는 디블록킹 필터는 8×8 단위로 적용되며 휘도인 경우 그림 6과 같이 한 개의 매크로 블록 당 수직, 수평 방향에서 각 4번의 필터링이 이루어진다.

그림 6에서 나타나는 BsV00, BsV10 등은 경계 세기(Boundary Strength)로 0부터 2까지 존재하여 숫자가 커질수록 강하게 필터링을 적용한다. Bs가 2일 때 저주파 통과 필터이며 필터 계수는 [1 2 1]/4 과 [2 1 1]/4 을 사용한다. 필터는 바꾸는 화소 개수에 따라 두 가지로 구분 될 수 있다. 다음은 그림 7을 참고하여 설명한

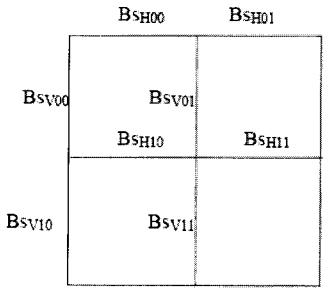


그림 6. 매크로 블록의 필터 적용을 위한 경계
Fig. 6. MB Boundary to be filtered.

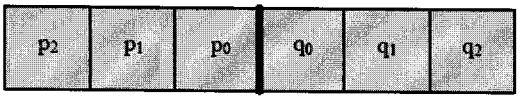


그림 7. 디블록킹 필터링을 위한 샘플 화소
Fig. 7. Sample pixels for deblocking filter.

것이다. 아래의 α, β값은 블록 내/간의 화소 값 차이에 대해 필터링 적용 여부를 결정하기 위한 임계값이다.

- abs(p2-p0) < β 이고 abs(p0-q0) < ((α>>2)+2)일 때 p1, p0, q0, q1을 바꾸며 p1, q1에 대해 [2 1 1]/4 계수를 적용하고 p0, q0에 대해 [1 2 1]/4 계수를 적용하여 필터링을 수행한다.
- 그렇지 않은 경우 p0, q0에 [2 1 1]/4 계수를 적용하여 필터링을 수행한다.

Bs가 1일 때에는 [1 -3 3 -1]/8의 계수를 가지는 고주파 통과 필터를 적용한다. 역시 필터링을 적용하는 방법이 두 가지로 구분되며 다음과 같다.

- p0 화소의 값은 p1, p0, q0, q1 화소에 필터링을 적용하여 p0값에 더한 값으로 바꾼다. q0은 필터링 된 값을 q0에서 뺀 값으로 바꾼다.
- abs(p2-p0) < β 일 때 p1 화소의 값은 p2, p1, p0, q0 화소에 필터링을 적용하여 p1 화소의 값에 더한 값으로 바꾼다. q1 화소의 값은 필터링 된 값을 q1 화소의 값에서 뺀 값으로 바꾼다.

III. 제안한 AVS 비디오 디코더

본 연구에서는 AVS 비디오 복호화기의 알고리즘 이해를 바탕으로 이를 DSP에 최적화 구현하였다. DSP에 최적화하기 위해 사용한 방법들은 다음과 같다. 전체적인 복호화기를 설계하여 구현하고 프로파일링을 통해 복호화기의 알고리즘 분석을 통해 메모리 사용량을 최소화 및 메모리 읽고 쓰는 연산을 감소시켰다. 또한, 프로파일링 결과 복호화기에서 연산량이 큰 디블록킹 필터에 대해 linear asm으로 구현하였으며 엔트로피 복호 성능을 올리기 위해 가변 길이 부호화 테이블을 설계하여 큰 성능 향상이 이루어졌다. SIMD의 구조를 효율적으로 사용하기 위한 intrinsic 적용, TI사의 컴파일러에서 제공되는 각종 pragma를 사용하여 큰 계산량 감소를 이루었다.

1. 재설계 및 프로파일링

AVS 비디오 복호화기는 H.264/AVC 비디오 복호화기에 비해 낮은 계산 복잡도의 알고리즘으로 이루어졌다. 그럼에도 불구하고 이를 DSP에 포팅하여 수행시, 낮은 클럭과 작은 메모리등의 하드웨어가 가지는 한계와 컴파일러가 Very Long Instruction Word (VLIW) 구조에 맞게 최적화 해 주지 못하는 소프트웨어가 가지는 한계로 인해 속도가 매우 느리다. 이러한 원인으로 펜티엄 4와 비교하였을 때 컴퓨터에 따라 다르지만 적어도 20배부터 50배 이상의 성능 차이가 나게 된다. 일단 AVS의 참조 소프트웨어인 RM의 경우 표준을 만족하지만 불필요한 메모리 사용이 많고 많은 사람들이 구현함으로써 코드의 효율성이 떨어진다. 따라서 본 연구에서는 각 모듈 및 구조체등을 정의하고 불필요한 코드를 사용하지 않는 등 새롭게 AVS 비디오 복호화기를 설계하고 이를 DSP에 포팅하였다.

설계된 비디오 복호화기를 TI의 통합 개발 환경인 CCS (Code Composer Studio)에서 제공하는 프로파일링 도구를 이용하여 QVGA급의 Foreman 영상을 QP 28로 부호화한 비트스트림으로 복호한 결과 각 기능 블록별로 보면 표 4와 같다. 표 4의 기능 블록별 연산량은 영상의 복잡도 및 QP값에 따라 차이가 발생할 수 있다.

설계된 AVS 비디오 복호화기는 복호 정보를 위한 메모리의 사용량을 줄이고 전체적인 코드 수정을 통해 구조를 개선하여 최적화 된 구조를 이루도록 하였으며 이 부분을 통해 RM 5.2j 복호화기와 비교하여 100% 가량의 성능 개선이 이루어졌다. 각 모듈별 최적화와 더불어 코드의 구조 개선이 중요하며 캐시 메모리를 효율적으로 사용하기 위해 프로그램 코드의 지역성 및 각 테이블의 지역성을 고려하여 코드를 분배하였다.

표 4. CCS에서의 프로파일링 결과
Table 4. Profiling result using CCS.

기능 블록	%
움직임 보간	40
역 변환	15
가변 길이 복호화	25
디블록킹 필터 및 기타	20

2. 메모리 사용 최적화

최적의 성능을 위해서는 프로그램의 전체 코드 부분을 내부 램에 로드하여 메모리 로드 시 발생하는 오버

헤드가 최소화 되어야 하지만 Davinci에서 제공하는 내부 램의 경우 64KB에 불과하다. 이 중에서 계층적인 메모리 구조를 위해 L2 캐시 메모리로 32KB를 사용하게 되는데 캐시 메모리의 특성상 자주 사용하는 블록에 대한 접근 속도가 향상되기 때문에 프로그램의 수행속도가 두 배 이상이 향상된다. 이와 같이 캐시 메모리의 양을 직접 결정하는 것은 DSP/BIOS (TI DSP에서 제공되는 Operating System)에서 설정할 수 있으며, 메모리의 크기 등을 변경하면서 테스트를 할 수 있다. DSP/BIOS 를 사용하여 빈번하게 사용되는 스택영역의 데이터에 대해 내부 램에서 수행될 수 있도록 설정을 할 수 있으며, 내부 램이 작기 때문에 제안한 디코더에서 스택영역으로의 메모리를 사용하는 경우 메모리의 할당이 크지 않도록 고려하여 설계 되었다. 또한, 캐시 메모리의 크기를 적절히 설정하는 것만으로도 성능이 올라가게 되지만 더욱 효율적으로 사용하기 위해서는 배열을 사용하거나 동적으로 메모리를 사용하게 될 때 캐시 메모리의 대역폭을 고려하여 생성을 해야 한다.

DSP에서 포인터나 배열 메모리를 함수 인자로 사용하는 경우 컴파일러는 컴파일 수행시 메모리 에일리어싱을 방지하기 위해 중복되지 않은 메모리 로드에도 불구하고 NOP (no operation)와 같은 코드를 넣게 된다. 하지만, 미리 컴파일러에게 그것을 알려줄 수 있다면 컴파일러의 불필요한 동작을 막을 수 있는데, DSP 컴파일러에서 제공하는 것은 restrict이라는 키워드이다. 함수에 인자로 배열이나 포인터를 넘겨주는 경우 restrict키워드를 사용하여 인자로 사용된 메모리 사이에는 서로 같은 주소 공간을 사용하지 않는다는 것을 컴파일러에게 알려줌으로써 컴파일러가 메모리 에일리어싱을 방지하기 위해 하는 동작을 줄일 수 있다.

3. Motion Compensation 모듈 최적화

프로파일링 결과 Motion Compensation (MC) 블록에서의 연산량이 가장 높게 측정이 되었으며, 이는 참조 영상에서 MC 과정을 수행시 보간 과정을 수행하고 블록 복사가 발생되기 때문이다. AVS에서는 1/2 및 1/4 위치의 화소에 대해서 4-tap 보간 필터를 사용한다. 정수 픽셀을 복사하는 경우에는 8 바이트의 메모리 연산이 가능한 _mem8() intrinsic을 사용하여 8×8 블록 복사를 8번의 반복문 수행만으로 처리하였으며 그 외에는 SIMD 구조로 연산을 수행하도록 하였다. 우선 1/2 화소 위치에 해당하는 경우 [-1 5 5 -1]/8 계수를 사용하

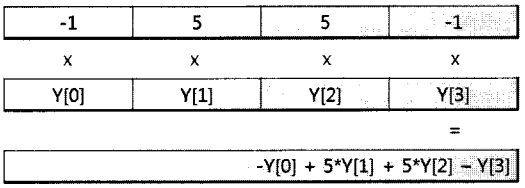


그림 8. 1/2 화소 위치의 보간법
Fig. 8. The Interpolation method for half pixel.

므로 그림 8과 같은 처리가 가능하다.

그림 8에서 나타낸 연산은 다음과 같은 방법에 의해 구현이 가능하다. 우선, 상수는 32 비트의 변수를 생성하여 0xff0505ff 와 같이 저장할 수 있다. 또한, 4 바이트를 동시에 로드 하는 _amem4 intrinsic을 이용하여 메모리에서 읽은 후 _dotpsu4()를 이용하면 위 연산이 이루어진다. 여기서 _dotpsu4()는 부호 있는 수와 부호 없는 수를 4 바이트 단위의 SIMD 명령어 처리를 통해 곱셈 후 그 결과를 모두 더해서 반환해 준다. 이와 같은 방법으로 1/4 화소 위치에 대해서도 처리를 하면 기존에 한 바이트씩 네 번을 메모리에서 읽은 후에 각각 곱하고 더하던 부분을 줄일 수 있다. 설명한 방법은 그림 9와 같은 코드로 표현되며 메모리 로드 및 곱셈을 1/4로 줄여 계산 복잡도를 줄일 수 있다.

위 방법으로 수직 방향의 경우에는 4 바이트씩 메모리에서 읽는 연산을 하는 것이 가능하지만 수평 방향에 대해서는 _dotpsu4() intrinsic으로 처리하는 것이 불가능하다. 따라서 본 연구에서는 그림 10에서와 같이 한 바이트를 수직으로 네 번 읽는 것을 반복문 바깥에서 처리하고 반복문 내에서는 한 번의 메모리 읽기를 수행

```
for (i = 0; i < 8; i++)
    for(j = 0; j < 8; j++)
        a1 = RefY[i-1][j];
        a2 = RefY[i  ][j];
        a3 = RefY[i+1][j];
        a4 = RefY[i+2][j];
        prediction[i*8+j] = CLIP(-a1+5*a2+5*a3-a4);
```

⇩

```
int const1 = 0xFF0505FF;
for (i = 0; i < 8; i++)
    for(j = 0; j < 8; j++)
        int a = _mem4(&RefY[i][j-1]);
        prediction[i*8+j] =_dotpsu4(const1, a);
```

그림 9. MC에서의 메모리 읽기 감소 1
Fig. 9. Memory read reduction in the MC module 1.

```
for (i = 0; i < 8; i++)
    for(j = 0; j < 8; j++)
        a1 = RefY[i-1][j];
        a2 = RefY[i  ][j];
        a3 = RefY[i+1][j];
        a4 = RefY[i+2][j];
        prediction[i*8+j] = CLIP(-a1+5*a2+5*a3-a4);

                ⇩

for (j = 0; j < 8; j++)
    a1 = 0;
    a2 = RefY[-1][j];
    a3 = RefY[ 0][j];
    a4 = RefY[+1][j];
    for(i = 0; i < 8; i++)
        a1 = a2;
        a2 = a3;
        a3 = a4;
        a4 = RefY[i+2][j];
        prediction[i*8+j] = CLIP(-a1+5*a2+5*a3-a4);
```

그림 10. MC에서의 메모리 읽기 감소 2
Fig. 10. Memory read reduction 2 in the MC module.

하여 성능을 향상시켰다. 또한 위 구조를 적용하기 위해 반복문 순회 순서를 행, 열에서 열, 행으로 순회하도록 수정하였다.

4. 가변 길이 복호화 테이블 설계 및 최적화

일반적으로 비트스트림을 파싱하는 비트 복호 과정의 경우 메모리에 있는 비트스트림에서 비트 단위로 파싱이 이루어져 연산량이 무척 큰 부분이다. AVS의 가변 길이 복호과정은 k 차수 지수-골룸 복호과정을 거치게 되는데, 그 중에서도 가장 연산량이 큰 부분은 텍스쳐 정보를 파싱하는 부분이다. 각 헤더 정보에 비해 텍스쳐 정보는 내용이 많고, 또한, 순차적인 구조로 메모리 접근이 이루어지기 때문에 DSP와 같은 구조에서는 더욱 성능을 떨어뜨리는 요인이다. 본 연구에서는 AVS의 텍스쳐 정보를 위한 가변 길이 복호화 테이블을 분석하고 디코딩 속도를 올리기 위한 테이블을 설계 하였다.

AVS비디오 코덱의 참조 소프트웨어인 RM 5.2J와 공개 소프트웨어인 OpenAVS에서는 가변 길이 복호화 테이블이 동일한 방식으로 구현되어 있다. AVS 표준의 테이블을 선언 후 연산을 통해 (run, level)을 참조할 수 있도록 새로운 테이블을 생성해 두는 것이다. 텍스쳐 정보는 지수-골룸 복호를 통해 생성된 값을 맵핑 테이블의 인덱스로 활용하여 (run, level) 값을 읽어 오는 방식이다.

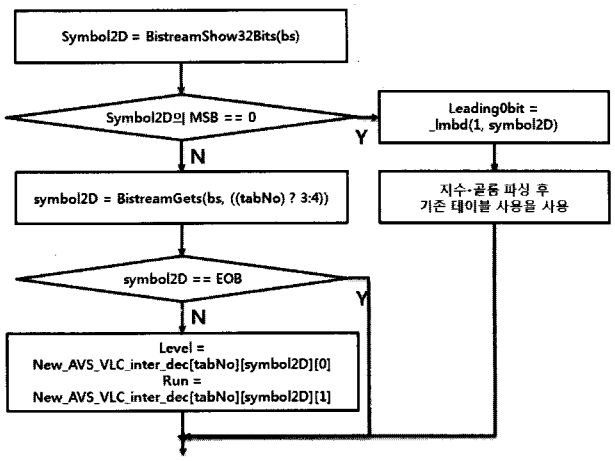


그림 11. 제안한 화면 간 예측을 위한 고속 VLD 알고리즘
Fig. 11. Block diagram of proposed fast VLD algorithm for inter MB.

제안된 화면 간 예측을 위한 가변 길이 테이블 복호화 방법은 다음과 같다. 일단, 표준 문서의 테이블은 실제 코덱에서는 사용할 필요없으므로 제거하고, 위에서 언급된 맵핑 테이블을 선언하여 사용한다. 제안된 방법은 맵핑 테이블에 추가적인 테이블을 사용하여 성능을 크게 향상 시켰다. 우선 비트스트림에서 현재까지 읽은 비트 오프셋을 보고 이후의 32비트를 읽어온다. 이 값을 symbol2D라고 하면 그림 11과 같이 제안된 알고리즘을 표현할 수 있다.

비트스트림으로부터 읽어온 32 비트의 symbol2D값에 대한 Most Significant Bit (MSB)를 확인 후 MSB가 1이면 제안한 방법을 사용한다. 여기서 MSB가 1인 경우는 그림 12에서 보듯이 k 차 지수-골룸 복호 과정에서 k 값과 상관없이 가장 상위의 값이라는 것을 의미한다. 예를 들어, k 값이 2일 때 MSB가 '1'인 경우는 상위 4개의 값 중에 하나이다. 따라서, 이 경우에는 MSB를 포함하여 세 비트의 정보만 읽어도 복호가 가능하다. 만일, k 값이 3인 경우에 MSB가 '1'이라면 상위 4비트를 읽음으로써 복호가 가능하다. 또한, 이 경우 제안한 새로운 가변 길이 복호 테이블을 사용하여 빠른 복호가 가능한데 8×8 블록에 존재하는 마지막 심볼이 아닌 경우에는 symbol2D값에 직접 맵핑되는 제안된 테이블에서 (run, level)을 가져온다. 여기서 말하는 테이블은 3 또는 4 비트를 비트스트림으로 읽어서 나오는 값을 인덱스로 하여 직접 맵핑될 수 있는 테이블이다.

Order	Code structure
k=2	100xxxxxx...
	101xxxxxx...
	110xxxxxx...
	111xxxxxx...
	0100xxxxxx...
	0101xxxxxx...
	0110xxxxxx...
	...
k=3	1000xxxxxx...
	1001xxxxxx...
	1010xxxxxx...
	1011xxxxxx...
	1100xxxxxx...
	1101xxxxxx...
	1110xxxxxx...
	...

그림 12. 차수가 2와 3일 때의 지수 골룸 복호 테이블
Fig. 12. Exp-Golomb tables of 2 and 3 order.

그 외에 MSB가 '0' 인 경우에는 테이블은 기존의 테이블을 사용하지만 _lmbd() intrinsic을 사용하여 '1'이 나올때까지 '0'의 개수를 세는 것을 통해 기존의 반복문 방식의 처리를 한 개의 명령어로 대체된다.

5. 디블록킹 필터의 Linear ASM 설계

DSP에는 VLIW 구조에 의해 A, B파트에 각각 L, D, S, M 유닛이 존재하여 총 8개의 유닛이 있다. L 유닛은 일반적인 논리 연산이나 시프트 연산 및 산술 연산을 담당하고, D 유닛은 메모리의 로드와 저장과 같은 메모리 연산과 산술 연산을 담당하며, S 유닛은 분기나 점프등과 산술 연산을 담당하고, M 유닛의 경우에는 일반 곱셈이나 SIMD 형태의 곱셈, 내적과 같은 연산을 담당하고 있다. 이처럼 각 유닛이 제공하는 연산은 조금씩 다르며 최적의 효과를 내기 위해서는 8개의 유닛이 병렬적으로 동시에 연산을 수행하는 것이 좋다. 또한, 각 유닛이 소프트웨어 파이프라이닝을 통하여 매 사이클 사용되면 좋은 성능을 낼 수 있는데, 이런 과정은 일반적으로 반복문 내에서만 가능하다. 하지만, 반복문 내에 분기문이 복잡하게 존재하는 경우 파이프라이닝이 잘 안 되게 되는데 이런 경우 분기문을 반복문 바깥에서 처리하거나 그림 13과 같이 산술 연산으로 대체하면 파이프라이닝을 유지할 수 있다.

```
for (i = 0; i < 8; i++)
    if (table_number == 3)
        table[table_number][i] = 5;
    else
        table[table_number][i] = 7;

                ⇩

for (i = 0; i < 8; i++)
    table[table_number][i] = 5 + (table_number != 3) * 2;
```

그림 13. 분기문 제거 예제
Fig. 13. Example of branch removal.

디블록킹 필터의 경우 경계 세기를 정하는 부분과 실제 필터링을 적용하는 부분으로 이루어져 있다. AVS의 경계 세기를 구하는 트리는 H.264/AVC의 결정 트리보다 간단하다. 즉, 화면 내 부호화의 경우 경계 세기는 2로 결정되며 화면 간 부호화에서는 다른 참조 영상이나 움직임 벡터가 다른 경우에만 경계 세기가 '1'이고 나머지 경우에 대해서는 '0'으로 필터링을 하지 않는다. 따라서 AVS의 디블록킹 필터 연산량은 대부분 필터링을 하는 부분에서 소요된다. AVS의 디블록킹 필터를 DSP에 구현한 기존 논문에서는 필터링을 하는 부분에 대해 6단계로 나누고, 이를 소프트웨어 파이프라이닝 구조로 DSP의 A, B 파트에 나눠서 구현하는 것을 제안하였다.[3] SIMD 명령어를 이용하여 데이터가 동시에 처리되는 부분은 없으나 DSP에서 제공하는 조건 분기문을 이용할 수 있도록 SUBABS4 명령어 및 비교 연산들을 사용하였다. 본 논문에서는 이 방법을 linear asm으로 적용하여 기존의 논문에서 설명하는 휘도 블록의 Bs가 1인 경우 외에 휘도, 색차 성분 각각 Bs가 1과 2인 모든 경우에 대하여 linear asm으로 구현을 하여 좋은 성능을 나타내고 있다.

6. Pragma 구문 사용

TI의 DSP 컴파일러에서 제공되는 pragma 문의 종류는 다양하다. 주로 사용되는 pragma 문은 반복문을 풀어서 쓰는 방법으로써 분기문을 줄이는 것이 가능한 UNROLL문, 반복문 반복 횟수의 최소, 최대 반복 횟수 등을 명시할 수 있는 MUST_ITERATE문, 메모리 영역을 사용자가 정할 수 있도록 해 주는 CODE_SECTION 문 및 DATA_SECTION문 그리고 배열을 생성할 때 alignment를 정해줄 수 있는 DATA_ALIGN문이 있다.

그 중에서도 본 연구를 위해 사용을 많이 한 것은 내/외부 메모리 관리를 위한 구문인 CODE_SECTION 및 DATA_SECTION이다. 내부 메모리의 효율적 사용을 위해 사용자가 사용할 수 있는 코드영역을 LINK 파일에 지정한 후 프로그램에 명시해 줌으로써 원하는 함수를 내부 메모리에 적재할 수 있다. 일반적으로 인라인 함수의 형태로 구현이 되어 있는 경우에는 코드가 실행 파일에 포함되므로 효과가 없으며 제안한 복호화기의 경우 매크로 블록 단위를 복호하는 함수 등에서 사용을 하여 메모리 사용을 효율적으로 하여 연산량을 감소시킬 수 있었다. 또한, DATA_SECTION을 이용하여 자주 사용되는 테이블을 내부 메모리에 올려 둘 수 있으며 DATA_ALIGN을 이용하여 배열을 이용 할 때 메모리를 다양한 바이트로 alignment하여 생성할 수 있다. 이것은 _nassert()라는 메모리의 alignment을 검사하는 함수와 더불어 사용을 하게 되어 컴파일러가 메모리 로드 및 저장을 수행 할 때 최적화를 할 수 있게 하여 성능을 높이게 된다. 그 외에 반복문을 풀어주는 구문인 UNROLL을 사용하여 반복문 구조를 사용자가 수정하지 않고 반복문 구조를 변경한 효과를 낼 수 있다. 또한, 컴파일러가 최적화 수행시 더 효율적으로 할 수 있게 MUST_ITERATE(min, max, count)를 넣어 반복문의 최소, 최대 반복 횟수에 대한 정보를 추가적으로 전달할 수 있다.

Ⅳ. 실험 결과 및 토의

본 논문에서 제안하는 최적화된 AVS 비디오 복호화기의 성능을 측정하기 위해서 참조 소프트웨어 RM 5.2j의 부호화기에서 4개의 영상에 대해 각각 3개의 QP 값으로 비트스트림을 생성하였다. 4개의 영상은 각각 D1 (720×480)급, QVGA (320×240)급 해상도의 영상들로 복잡도가 서로 다른 영상들이며 표 5에서 테스트 영상에 대한 스펙을 자세하게 나타내었다. 제안한 복호화기는 C64x+ 계열의 하나인 DM6446 EVM에 최적화하였다. 또한, 실험 결과의 객관적인 비교를 위해 RM 5.2j의 복호화기를 위한 실험을 동일한 보드에서 진행하였다. DM6446 EVM보드는 594Mhz의 클럭 주파수를 가지고 있으며 L1D 및 L1P의 캐시 메모리가 각각 32K 존재한다. L2 캐시 메모리 및 내부 램으로 사용할 수 있는 SRAM이 64K 존재하는데 이번 실험을 위해서는 L2 캐시 메모리 32K, 내부 메모리 32K씩을 사용하였다. 컴파일러 옵션은 -o3만을 사용하였는데 -o3는 컴파일러 최적화 옵션 중 하나이다. 일반적으로 사용하는 프로그램 레벨에서의 최적화를 해 주는 -pm op2의 경우에는 적용하는 경우 전체 속도가 저하되어 사용하지 않았다.

표 5. 테스트 영상 명세
Table 5. Specification of test video clips.

	D1 (720×480)	QVGA (320×240)
영상	Mother and Daughter, Susie	Football, Container
QP	25, 30, 35	25, 30, 35
부호화기	RM 5.2j 부호화기	
GOP	IPPP 구조	

표 6. RM 5.2j 복호화기와 제안한 복호화기의 VLD 속도 비교
Table 6. VLD performanece of the proposed method compared with that of RM 5.2j.

Sequence	QP	Cycle		향상율
		RM 5.2j	Proposed	%
Football	25	614,099,437	61,703,079	895.25
	30	502,879,438	42,478,577	1083.84
	35	416,619,417	29,005,374	1336.35
Container	25	278,134,455	11,398,734	2340.05
	30	161,479,482	6,046,837	2570.48
	35	95,762,786	3,213,064	2880.42
Mother & daughter	25	1,664,127,488	50,251,315	3211.61
	30	647,154,946	15,766,688	4004.57
	35	417,474,765	9,419,169	4332.18
Susie	25	2,183,838,040	96,834,748	2155.22
	30	1,383,746,356	43,261,200	3098.59
	35	915,212,126	24,009,829	3711.82
평 균				2635.03

실험은 두 가지로 진행을 하였다. 가장 크게 성능을 향상 시킨 요인 중 하나인 VLD의 속도 향상에 대한 실험과 제안한 AVS 비디오 복호화기의 전체 복호 속도에 대한 실험이다. VLD의 속도 향상에 대한 실험 결과는 표 6, 그리고 전체 복호화기의 속도에 대한 실험 결과는 표 7에 정리하였다.

표 6에서의 VLD 성능을 클럭의 차이를 이용하여 비교해 보면 평균적으로 26배 가량의 속도 향상이 이루어진 것을 볼 수 있다. 고속의 VLD를 위한 테이블의 사용으로 RM의 VLD 복호 성능과 비교하여 5~15배 이

표 7. RM 5.2j 복호화기와 제안한 복호화기의 전체 속도 비교

Table 7. Decoding performance of decoder using the proposed method with that of RM 5.2j.

Sequence	QP	초당 복호 프레임 수		향상율
		RM 5.2j	Proposed	%
Football	25	10.45	64.05	512.92
	30	11.87	72.92	514.32
	35	13.19	82.36	524.41
Container	25	16,82	121.48	622.24
	30	20.53	151.32	637.07
	35	23.59	181.76	670.50
Mother & daughter	25	3.60	31.04	760.37
	30	5.59	44.88	701.57
	35	6.42	51.46	700.82
Susie	25	3.28	23.23	606.35
	30	4.21	29.06	589.82
	35	4.80	33.39	594.32
평 균				619.55

르는 복호 속도 향상이 이루어 졌으며 또한, 테이블의 선택과 관련한 분기문들을 산술 연산으로 대체함으로써 약 두 배의 속도 향상이 이루어졌다. 특히, QP 값이 클수록 복호를 하기 위한 VLD 모듈의 계산량이 증가하므로 제안한 고속의 가변 길이 복호 방법이 효율적이다. 또한, QVGA급 영상보다 D1급 영상에서 더 좋은 성능이 나오고 있으며, 이에 따라 AVS 비디오 코덱이 주요 타겟으로 하는 VGA급 이상에서의 수행 시 더욱 높은 평균값을 얻게 될 것이다.

표 7은 제안한 방법의 전체적인 성능 차이를 RM 복호화기와 비교하여 나타내고 있다. AVS 복호화기의 재설계를 통해 기존의 RM 복호화기와 비교하여 약 두 배의 속도 향상이 이루어졌으며 제안한 VLD 알고리즘과 산술 연산 최적화로 역시 두 배가 향상 되었다. 디블록킹 필터의 linear asm 구현과 MC에 대한 intrinsic을 적용하여 약 50%의 속도 향상으로 제안한 방법이 RM 복호화기보다 평균 6배의 복호 속도 향상을 보이며 여러 가지 영상 크기 및 QP 값에 대해서도 유사한 결과를 보인다. RM 5.2j의 경우 QVGA급 영상을 복호 하는 경우에도 실시간으로 동작하지 않을 정도로 느리지만 제안한 복호화기를 이용하여 D1급 영상에서 실시간에 가까운 복호가 이루어진다.

V. 결 론

본 논문에서는 중국에서 각종 멀티미디어 응용 사업에 사용되고 있는 AVS 비디오 복호화기의 알고리즘을 설명하고 고속의 AVS 비디오 복호화기를 제안하여 DSP에서 좋은 결과를 나타내었다. 알고리즘의 이해를 통한 구조적인 최적화 외에도 DSP에서의 intrinsic, linear asm 및 컴파일러의 성능을 올릴 수 있는 도구들을 구현함으로써 복호화기의 수행 속도를 높였다. 성능 검증을 위해 AVS의 참조 소프트웨어인 RM 5.2j과 제안한 AVS 비디오 복호화기에 대해 동일한 비트스트림을 이용하여 DSP에서 실험함으로써 비교하였다. 실험 결과 분석을 통해 참조 소프트웨어에 비해 크게 향상된 성능을 보이는 것을 확인하였다. 앞으로의 연구방향은 각 모듈간의 메모리 사용을 더욱 줄이고 세부 모듈에 대해 저 레벨의 구현을 함으로써 더욱 향상된 복호화기의 성능을 갖도록 하는 것이다.

참 고 문 헌

[1] "http://radar.ndsl.kr", GTB2007080150, August 7, 2007.

[2] "http://radar.ndsl.kr", GTB2006110606, November 12, 2006.

[3] Zhigang Yang, "DSP implementation of deblocking filter for AVS," in Proc. *ICIP*, vol. 6, pp. 205-208, 2007.

[4] AVS workgroup, "Reference software version RM5.2j"

[5] "*TMS320C6000 Programmer's Guide (Rev.I)*," 2006, http://www.ti.com

[6] "*TMS320C6000 C64x+ DSP CPU and Instruction Set Reference Guide*," SPRU732A, Jun. 2005, http://www.ti.com

[7] "*TMS320C64x+ DSP Cache User's Guide (Rev. A)*," 2007, http://www.ti.com

[8] "*Variable-Length Decoding on the TMS320C6000 DSP Platform*," 2005, http://www.ti.com

[9] GB/T 20090.2-2006 "Information Technology Advanced Audio and Video Coding Part 2:Video: AVS-P2"

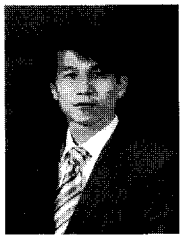
[10] JVT, "*Draft ITU-T Recommendation H.264 and Draft ISO/IEC 14496-10 AVC*," Doc JVT-50, Thailand, Mar. 2003.

[11] 강대범, 황정우, 심동규, "TMS320C64x+를 이용한 MPEG-4 코덱 최적화," *제 20회 신호처리합동학술대회논문지*, 1권, 158쪽, 2007년 10월

[12] 강대범, 심동규, 박호종, 심영석, "Davinci를 위한 Sorenson H.263 비디오 디코더 최적화," *제 21회 신호처리합동학술대회논문지*, 1권 155쪽, 2008년 9월
[13] openavs-1.0.3, "http://sourceforge.net", July 7, 2006.

저 자 소 개

강 대 범(학생회원)
2008년 광운대학교 컴퓨터공학과 학사 졸업.
2008년~현재 광운대학교 컴퓨터 공학과 석사과정.
<주관심분야 : 영상신호처리, 영상 압축, DSP, ASIP>



심 동 규(정회원)
1999년 서강대학교 전자공학과 공학박사.
1999년~2000년 (주) 현대 전자.
2000년~2002년 (주) 바로 비젼.
2002년~2005년 Univ. of Washington
2005~현재 광운대학교 컴퓨터공학과 (부교수)
<주관심분야 : 영상신호처리, 영상압축, 컴퓨터 비젼>