H.264/AVC응용을 위한 Exp-Golomb CODEC의 설계

김원삼* · 손승일*

*한신대학교 정보통신학과

Design of Exp-Golomb CODEC for H.264/AVC Applications

Won-sam Kim* · Seung-il Sonh*

*Department of Information and Communication, Hanshin University

E-mail : ryuna80@gmail.com, saisonh@hs.ac.kr

요 약

가변길이 부호는 많은 이미지 및 영상 표준에서 폭넓게 사용되는 기법이다. 특히 국제 표준인 JVT와 중국 A/V 표준인 AVS는 엔트로피 코딩을 수행하기 위해 Exp-Golomb 코드에 기반한 UVLC(Universal Variable Length Code)를 채용하고 있다. 본 논문에서는 H.264/AVC의 엔트로피 코딩에서 사용되는 Exp-Golomb CODEC의 하드웨어 구현에 대해 연구하였다. 식의 간략화로 구현하기 어려운 log함수와 거듭제곱 연산을 하지 않으며, 첫 번째 1 검출기와 누산기 제어에 의한 배럴 쉬프터를 통하여 별도의 시간 지연 없이 부호화 및 복호화가 되도록 설계하였다. Xilinx ISE툴을 사용하여 합성하고, 보드 수준에서 PCI인터페이스를 사용하여 검증하였다. 본 논문에서 설계된 Exp-Glomb CODEC은 H.264/AVC 및 AVS와 같은 분야에서 응용이 가능할 것으로 예견된다.

키워드

H.264/AVC, UVLC, Exp-Golomb Code, CODEC

Ⅰ. 서 론

ISO/IEC MPEG 그리고 ITU-T VCEG의 Joint Video Team(JVT)은 비디오 영상들의 부호화를 위해 새로운 표준을 제안하였다. 새로운 표준은 H.264 혹은 MPEG-4 part 10, "Advanced Video Coding"으로 부른다. 이 표준의 엔트로피 코딩은 슬라이스 레이어 상위에서는 문맥적 요소들(Syntax elements)이 고정길이 또는 가변길이 이진 코드로 인코딩된다. 여기서, entropy_coding_mode가 0이면 오차 블록 데이터는 컨텍스트 적응형 가변길이 코딩(Context-based Adaptive Variable Length Coding, CAVLC)을 사용하여 코딩되고 기타 가변 길이 코딩된 단위들은 Exp-Golomb 코드로 코딩된다[1][2]. 본 논문에서는 매크로블록의 종류, 참조 프레임 인덱스, 움직임 벡터 차이 등에 사용되어지는 Exp-Golomb 인코더 와 디코더를 설계하였다.

Ⅱ. Exp-Golomb 코딩

Exp-Golomb 코드(Exponential Goloms codes)는 규칙적인 구조를 가지는 가변길이 코드이다. 표는 비트 스트링(bit string)을 "prefix"와 "suffix" bits로 분리함으로써 Exp-Golomb code의 구조를 나타낸다. "prefix" bits는 표에서 0 혹은 1로 보여지는 비트이다. "suffix" bits는 xi로 보여진다. xi는 0 혹인 1의 값을 가질 수 있다[1][3].

[표 1] Exp-Golomb 코드의 구조

code_num의 범위	코드워드 형태
0	1
1-2	0 1 x_1
3-6	0 0 1 x_1 x_0
7-14	0 0 0 1 x_2 x_1 x_0
15-30	0 0 0 0 1 x_3 x_2 x_1 x_0
...	...

[M zeros][1][INFO] (식 1)

각 Exp-Golomb 코드워드의 길이는 (2M+1) 비트이고 각 코드워드는 인코더에서 그 인덱스인 code_num을 기반으로 구성된다.

$$M = floor(\log_2[code\_num + 1]) \quad (식\ 2)$$

$$INFO = code\_num + 1 - 2^{M} \quad \text{(식 3)}$$

표 2는 code_num에 따른 코드워드를 보여준다.

[표 2] Exp-Golomb codewords

code_num	Codeword
0	1
1	010
2	011
3	00100
4	00101
5	00110
...	...

unsigned Exp-Golomb codes에 대한 매핑 과정은 code_num을 입력으로 하고 출력은 ue(v)로 부호화된 문맥적 요소의 값이다.

code_num = syntax element (식 4)

signed Exp-Golomb codes에 대한 매핑 과정은 code_num을 입력으로 하고 출력은 se(v)로 부호화된 문맥적 요소의 값이다. code_num을 구하는 식은 아래와 같다.

$$code\_num = 2|v| \quad (v \le 0) \quad \text{(식 5)}$$

$$code\_num = 2|v| - 1 \quad (v > 0) \quad \text{(식 6)}$$

다음의 표 3은 signed mapping을 나타낸다.

[표 3] Signed mapping se

code_num	syntax element value
0	0
1	1
2	-1
3	2
4	-2
5	3
k	$(-1)^{k+1}$Ceil(k/2)

표 4는 매크로블록 예측 모드가 Intra_4×4 인지 혹은 Inter인지에 따라 coded_block_pattern에 할당된 code_num의 일부를 보여준다.

III. 제안하는 Exp-Golomb 인코더의 구현

[표 4] cbp에 할당된 code_num

code_num	code_block_pattern	
	Intra_4×4	Inter
0	47	0
1	31	16
2	15	1
3	0	2
4	23	4
...	...	...
47	41	41

ue방식의 문맥적 요소는 식 4와 같이 code_num과 같기 때문에 1을 더해주면 prefix를 제외한 '1' & INFO를 구할 수 있다.

$$code\_num + 1 \simeq v + 1 \quad \text{(식 7)}$$

위 식에서 '≃' 기호는 값은 같지만 비트수가 틀림을 의미한다.

se방식의 문맥적 요소는 0 보다 클 때 식 6을 통하여 code_num을 구해 1을 더하면 '1' & INFO를 구할 수 있으며 아래의 식의 전개를 통하여 간단한 왼쪽 쉬프트 연산으로 구할 수 있다.

$$\begin{aligned} code\_num + 1 &\simeq 2|v| - 1 + 1 \\ &= 2|v| \\ &= v \ll 1 \end{aligned} \quad \text{(식 8)}$$

syntax element가 0 보다 작거나 같을 때 식 5를 통하여 code_num을 구하여 1을 더하면 '1' & INFO를 구할 수 있으며 식 9의 전개로 연산수를 줄일 수 있다.

$$\begin{aligned} code\_num + 1 &\simeq 2|v| + 1 \\ &= (\bar{v} + 1) \ll 1 + 1 \\ &= \bar{v} \ll 1 + 2 + 1 \\ &= \bar{v} \ll 1 + 3 \end{aligned} \quad \text{(식 9)}$$

me방식의 문맥적 요소는 표 5와 같이 0~47 사이일 때 각 값에 따른 테이블을 참조하며 그렇지 않으면 ue방식과 똑같이 처리를 한다. 또한 테이블에 출력 데이터를 +1씩 더해서 저장을 했기 때문에 굳이 increment를 하지 않아도 되도록 설계하였다.

16비트의 code_num+1을 구하면 여기서 유효한 '1' & INFO는 그대로 유지시키되 "prefix" 인 M개의 0의 수를 구해 0~31 비트의 codeword로 만들어 줘야 하기 때문에 첫 번째 '1'을 찾는 것은 매우 중요하다. 첫 번째 1을 찾기 위해서 Wu Di가 제안한 첫 번째 1 검출기[4]를 사용하였으며 블록도는 그림 1과 같다.

이 검출기에서 각각의 OR게이트 출력은 1의

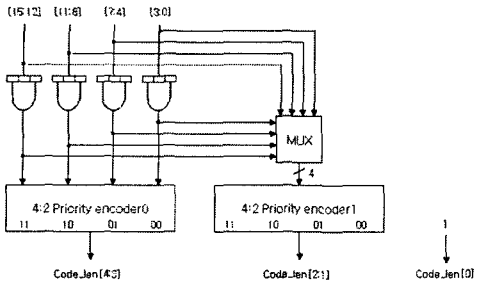

그림 1 첫 번째 1 검출기

존재여부를 알려준다. 이 출력은 4:2 Priority encoder0 에 입력되며 우선순위 인코더를 통하여 출력되어지는 11, 10, 01, 00은 12, 8, 4, 0을 의미한다. 오른쪽 멀티플렉서는 OR게이트의 출력을 통해 제어되어 첫 번째 1이 있는 4비트를 출력시키며 4:2 Priority encoder1 또한 11, 10, 01, 00을 출력시키며 이는 3, 2, 1, 0을 의미한다. 이렇게 출력된 값은 첫 번째 1이 최하위 비트로부터 몇 번째 비트인지, 즉 suffix의 길이를 의미하며 코드워드의 길이를 구하기 위해 왼쪽으로 한번 쉬프트하고 1을 더해줘야 한다.

codeword의 길이를 통하여 16비트의 전처리 모듈의 출력은 배럴 쉬프터에 의하여 왼쪽 쉬프트를 하게 된다. 배럴 쉬프트에 의하여 유효한 코드워드의 길이는 최대 31비트인데 전체 모듈의 출력이 32비트 이상이 유효해야 출력으로 나갈 수 있으므로 이전에 코드워드로 인코딩이 되었지만 출력되지 못하고 저장되어 있을 수 있는 최대 길이 또한 31비트이다. 따라서 64비트의 배럴 쉬프터를 코드의 길이에 따라 제어가 되도록 구현을 하였으며 전체 모듈은 그림 2와 같다.

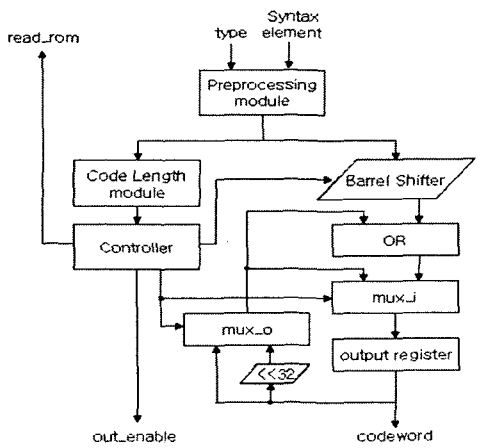

그림 2 제안하는 Exp-Golomb 인코더

Ⅳ. 제안하는 Exp-Golomb 디코더의 구현

하나 이상의 코드워드로 이루어진 32bit의 비트스트림이 입력되어 지면 순차적으로 개개의 코드워드를 분리시키는 것이 필요하기 때문에 저장되어야 할 레지스터가 필요하다. 게다가 32bit의 최하위에 존재하는 코드워드는 다음에 입력되어지는 32bit의 최상위에 연결되어 있을 가능성이 있으므로 32bit의 레지스터 2개가 존재해야 한다.

첫 번째 입력된 32bit는 그대로 Barrel shifter0을 통하여 코드 길이 모듈과 Barrel shifter1에 입력된다. 코드 길이 모듈은 인코더의 모듈과 동일하며 이 출력은 첫 번째로 검출된 codeword를 제외한 나머지 비트들의 수를 의미한다. 이 수에 의해 Barrel Shifter1에 우측 쉬프트를 제어하고 최하위 16비트가 출력되어진다. 이것은 최상위 16비트는 모두 0이기 때문이다.

코드 길이 모듈의 출력을 32에서 빼게 되면 현재 코드워드의 길이를 의미하며 이 값은 누산기에 입력되어진다.

다음 클록에는 누산기에 의해 디코딩된 코드워드를 제외한 나머지 32비트가 Barrel Shifter0에서 출력되어져 순차적으로 각각의 codeword를 분리해나간다. 누산기에 값이 32bit 이상이 되어 캐리가 발생하였다는 것은 R1 레지스터가 모두 디코딩이 되었다는 의미이고 그 다음 클록에 R0 데이터는 R1으로 입력되어지고 새로운 32bit의 데이터가 R0에 입력되어진다. 이것을 구현한 블록도는 그림 3과 같다.

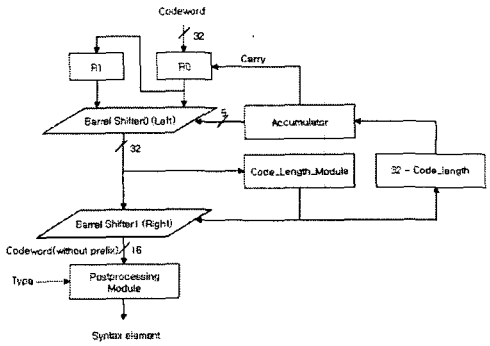

그림 3 제안하는 Exp-Golomb 디코더

Barrel shifter1의 출력은 codeword의 suffix를 제외하면 code_num+1과 같으므로 각각의 방식에 따라 후처리를 하면 문맥적 요소를 구할 수 있다.

ue방식일때는 식 10을 통하여 쉽게 구할 수 있다.

$$v \simeq (code\_num + 1) - 1 \qquad \text{(식 10)}$$

se방식은 code_num의 패리티에 의존을 한다. 표 3에 나와 있는 식을 살펴보면 code_num이 홀수라면 code_num+1은 항상 짝수이기 때문에

(-1)k+1은 항상 1이 되어 의미가 없어진다.

$$v \simeq (code\_num + 1) \gg 1 \qquad \text{(식 11)}$$

만약 code_num이 짝수라면 나머지가 없기 때문에 Ceil함수는 의미가 없어져 식 12와 같이 나타낼 수 있다.

$$v \simeq \overline{(code\_num \gg 1)} + 1 \qquad \text{(식 12)}$$

me방식은 인코더와 마찬가지로 1~48사이의 입력이 들어오면 각각에 해당하는 문맥적 요소를 출력하며 그렇지 않은 경우에는 ue방식으로 처리한다.

V. 결 론

본 논문에서는 인코더에서 code_num+1을 구하는 과정과 디코더에서 문맥적 요소를 구하는 과정의 식을 간략화 하고 me방식에서 테이블을 통하여 미리 +1이 더해져 있는 수를 참조함으로써 표 5~6과 같이 하드웨어의 면적을 줄일 수 있도록 설계하였다. 우선순위 인코더를 이용한 코드 길이를 구하는 모듈과 누산기를 이용하여 배럴 쉬프터를 제어하는 모듈을 통해 인코더에서는 한 클록마다 하나의 codeword를 내부에서 생성하여 32비트 되었을 때 출력 하도록 하였고 디코더에서는 한 클록 마다 하나의 문맥적 요소가 출력이 되도록 하였다.

[표 5] Exp-Golomb 인코더의 면적

모듈	게이트 수
배럴 쉬프터	1,401
첫 번째 1 검출기	122
FSM	165
전처리 모듈	818
인코더 전체 모듈	4,106

[표 6] Exp-Golomb 디코더의 면적

모듈	게이트 수
배럴 쉬프터0	1,626
배럴 쉬프터1	939
첫 번째 1 검출기	93
누산기	178
후처리 모듈	574
FSM	34
디코더 전체 모듈	5,863

구현된 인코더 및 디코더 모듈들은 PCI 인터페이스를 통한 Xilinx VirtexE FPGA 구성을 이용하여 합성하고 검증하였다.

이는 H.264/AVC를 채택하여 실시간과 소형화를 추구하는 동영상 서비스 응용분야에 적합할 것으로 사료된다.

참고문헌

[1] 호요성, 김승환, "H.264/AVC 표준의 소스 코드 분석," 두양사, 2006. 7.

[2] Richardson, Iain E. G, "H.264 and Mpeg-4 Video Compression," John Wiley & Sons Inc, Dec. 2003.

[3] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, May 2003, Joint Video Team.

[4] Wu Di, Gao Wen, Hu Mingzeng, and Ji Zhenzhou, "An VLSI Architecture Design of CAVLC Decoder," ASIC, 2003. Proceedings. 5th International Conference Vol.2, OCT 2003, pp.962~ 965.