논문 2007-44SP-4-7

HD급 영상을 효율적으로 복호하기 위한 CAVLC 복호화기 VLSI 설계

(Efficient CAVLC Decoder VLSI Design for HD Images)

오 명 석*, 이 원 재*, 김 재 석**

(Myungseok Oh, Wonjae Lee, and Jaeseok Kim)

요 약

본 논문에서는 H.264/AVC 엔트로피 부호화기의 베이스라인(Baseline)과 익스텐디드(Extended) 프로파일에서 사용되는 내용 기반 가변 길이 부호화(CAVLC: Context-based Adaptive Variable Length Coding)의 하드웨어 기반 복호화기 구조를 제안한다. 기존에 제안되었던 CAVLC 복호화기 하드웨어 구조는 5단계의 블록으로 설계되어 있고, 각 블록들이 유효비트를 얻기 위해서는 컨트롤러블록과 Accumulator블록을 거쳐 구해진다. 이때 레지듀얼 계수가 많을수록 이 과정을 여러 번 반복하게 되기 때문에 복호화 효율이 떨어진다. 본 논문에서는 이러한 유효비트를 구하는 과정을 줄이기 위해 2가지 방법을 제안한다. 한 가지 방법은 5단계로 이루어져 있던 블록을 4단계의 블록으로 줄이는 것이고, 다른 한 가지 방법은 컨트롤러에 의한 덧셈 연산 단계를 생략함으로써 블록별 유효비트를 효율적으로 구하는 것이다. 제안된 방법을 적용한 구조에 실험한 결과, 하드웨어의 크기는 비슷하면서 하나의 매크로블록을 처리하는데 요구되는 평균 사이클 수가 기존의 방식보다 약 26% 줄었고 0.18um 표준 셀 라이브러리로 합성한 결과 14.2K 게이트를 가졌다.

Abstract

Abstract– In this paper, we propose an efficient hardware architecture for H.264/AVC CAVLC (Context-based Adaptive Variable Length Coding) decoding which used for baseline profile and extended profile. Previous CAVLC architectures are consisted of five step block and each block gets effective bits from Controller block and Accumulator. If large number of non-zero coefficients exist, process for getting effective bits has to iterates many times. In order to reduce this unnecessary process, we propose two techniques, which combine five steps into four steps and reduce process to get efficiency bit by skipping addition step. By adopting these two techniques, the required processing time was reduced about 26% compared with previous architectures. It was designed in a hardware description language and total logic gate count was 16.83k using 0.18um standard cell library.

Keywords: H.264/AVC, Entropy coding, CAVLC, CAVLD, Baseline entropy coding

I. 서 론

H.264/AVC는 ISO/IEC의 Moving Picture Experts Group과 ITU-T의 Video Coding Experts Group이 공동 연구를 수행하여 만든 동영상 압축 국제 표준이다. 이 표준은 이전의 동영상 압축 표준과는 달리 높은 압축 효율과 네트웍 친화적인 서비스 제공을 목적으로 기존 하이브리드 부호화 구조에 1/4화소 단위의 움직임 추정, 인트라 추정, 정수 변환, 루프 필터와 컨텍스트 기반 엔트로피 부호화 기법을 적용하였다. H.264/AVC의 압축효율은 이전 동영상 표준보다 2배 이상 향상되었으나, 복잡도가 최대 16배 이상 증가되어 이를 실시간 처리하기 위한 하드웨어 기반의 구조 설계가 요구되고 있다.

H.264는 변환계수를 위하여 CAVLC를 사용하며, 다른 구문 요소(syntax elements)들을 위하여 Exp-

* 학생회원, ** 정회원, 연세대학교 전기전자공학과
(Dept. Electrical and Electronic Eng. Yonsei Univ.)
※ 본 연구는 정보 통신부 및 정보통신연구진흥원 대학 IT연구센터 육성·지원사업 및 2006년도 교육인적자원부 BK21 사업의 일환인 연세대학교 전기전자공학부 TMS 사업단의 지원을 받아 연구되었고, CAD Tool은 IDEC으로부터 지원 받았음.
접수일자: 2007년4월3일, 수정완료일: 2007년6월4일

Golomb 코드 부호화 방식을 사용한다. 이전 표준에서의 변환계수코딩을 하기 위해서는 순방향 zig-zag scanned run-length coding과 fixed variable length coding(VLC)기법을 사용하였으나 압축 효율을 높이기 위해 역방향 zig-zag scanned run-length coding과 adaptive VLC가 이용되었다. 이 CAVLC는 total_coeff, trailing_ones, level, total_zero, run_before 등의 5단계로 이루어져 있다[1]. 각 단계는 가변적인 길이의 비트를 이용하여 복호하기 때문에 이전의 단계가 복호되기 전에 다음 단계를 수행할 수 없다. 따라서 각 단계는 순차적으로 복호를 진행하게 되며 이때 각 단계별 복호하는데 필요한 유효비트를 빠르게 알아내는 것은 CAVLC의 복호를 더욱 효율적으로 진행하는데 필요하다. 룩업테이블을 효율적으로 복호하는 방식과 zero_skip 방식은 각 단계를 복호한 후 복호된 코드워드의 길이를 컨트롤러에 보낸 후 컨트롤러에서 적절한 값을 Accumulator에 저장한 후 그 값만큼 배럴쉬프터를 shift함으로써 원하는 유효비트를 구할 수 있었다[2-3]. 그러나 이와 같은 방식으로 배럴쉬프터를 shift할 경우 각 단계별 과정을 수행하기 전에 유효비트를 구하기 위해 많은 수행시간이 요구된다.

본 논문은 위와 같은 유효비트를 구하는 수행시간을 줄이기 위해 두 가지 알고리즘을 제안한다. 첫 번째, 컨트롤러와 Accumulator를 이용하여 유효비트를 구하는 단계를 대신에 기존 구조에 64비트 쉬프트 레지스터를 추가로 사용하여 단계별 유효비트를 효율적으로 구하는 방법과 두 번째, Total_coeff블록과 Trailing_ones 블록의 두 단계를 하나의 단계로 처리하는 방법이다. 이 두 가지 알고리즘은 유효비트를 구하는 수행시간 뿐 아니라 전체 CAVLC 수행시간을 줄일 수 있다. 제안된 알고리즘은 Verilog HDL을 이용하여 H/W로 구현 및 검증되었다.

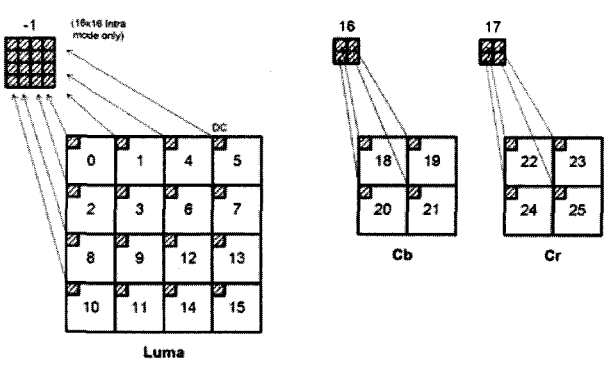

그림 1. 매크로블록의 코딩 순서
Fig. 1. Coding order of Block in a Macroblock.

본 논문의 구성은 다음과 같다. 먼저 Ⅱ장에서는 CAVLC의 복호화 과정 및 기존 알고리즘에 대하여 살펴보고 Ⅲ장에서는 유효비트를 구하기 위해 소요되는 클록을 줄이기 위한 알고리즘을 제시한다. 그리고 Ⅳ장에서는 기존에 제시된 구조와 비교분석을 하고, 마지막 Ⅴ장에서는 본 논문의 결론을 맺는다.

Ⅱ. CAVLC 복호화

2.1. CAVLC 복호화 과정

베이스라인과 익스텐디드 프로파일에서 정수 변화된 레지듀얼 계수는 CAVLC로 부호화 하고 움직임 벡터와 양자화 값 등의 다른 구문요소들은 Exp-Golomb 코드로 부호화 한다. 따라서 CAVLC는 시간적 중복성과 주파수적 중복성이 제거된 블록의 계수들을 무 손실 압축하기 위한 알고리즘이다. H.264에서 레지듀얼 계수의 부호화 단위는 16화소x16화소 단위의 매크로 블록이며, 매크로 블록은 4화소x4화소 단위의 휘도 DC 블록, 휘도 AC 블록, 채도 AC 블록, 2화소x2화소 단위의 채도 DC 블록으로 구성되어 있다. 이 블록의 화소 값들은 CAVLC 알고리즘으로 복호화 되고 복호화순서는 그림 1과 같다. 각 블록의 번호는 복호화 순서를 나타내며, 이순서는 매크로블록 타입과 CBP(Coded Block Pattern)에 의해 결정되어진다. 인트라 16화소x16화소 모드로 인트라 추정되었을 경우 휘도 DC블록이 존재하기 때문에 CAVLC는 블록번호 -1부터 순서대로 진행되며 그렇지 않을 경우(인트라 4화소x4화소 모드) 블록번호 0부터 진행된다. 그리고 휘도 및 채도 블록에 할당되어 있는 CBP값의 활성화 여부에 따라 블록 부호화를 조건적으로 수행하고 다음 CBP값에 해당되는 블록을 부호화하여 부호화 효율을 높이도록 구성된다[4].

CAVLC는 위와 같은 순서로 채도 DC는 2화소x2화소, 그 외는 4화소x4화소 단위로 레지듀얼 계수를 복호화 한다. 다음은 CAVLC의 복호화 5단계를 설명한다.[5]

1. 블록내의 0이 아닌 계수의 개수(total_coeff)와 ±1의 개수(trailing_ones)를 복호화 한다. 이 값들을 복호하기 위해 룩업 테이블을 이용한다. 이 테이블은 주변 블록에 따라 5개의 테이블로 나누어지고 주변 블록에 따라 적절한 테이블을 참조하기 위해서는 현재 복호하는 블록의 왼쪽(nA)과 위쪽(nB) 블록의 total_coeff을 이용한다.

2. 0이 아닌 계수들 중에서 ±1의 부호를 복호화 한다.

비트의 값이 1일 경우에는 -1로 0일 경우에는 +1로 복호화 한다. trailing_ones의 개수는 3을 넘을 수 없기 때문에 3번째 이후의 ±1 계수들은 다음 단계인 레벨에서 복호된다.

3. 0이 아닌 계수들 중에서 trailing_ones를 제외한 나머지 계수(level)들을 복호한다. 따라서 레벨의 개수는 total_coeff에서 trailing_ones의 수를 뺀 값이 된다. 레벨 값들은 zigzag scan의 역순으로 복호화가 진행된다. 레벨의 값들을 복호하기 위한 7가지의 룩업 테이블이 있다.

4. 마지막 계수 이전의 0의 개수인 total_zero를 복호한다. total_zero는 total_coeff와 입력 비트에 대한 룩업 테이블을 참조한다. 채도 DC 2화소x2화소에 대한 룩업 테이블과 그 외의 4화소x4화소 블록을 복호하는 룩업 테이블로 구성되어 있다.

5. 0이 아닌 계수들 사이에 0의 개수 run_before를 복호한다. run_before는 룩업 테이블을 사용하여 각 계수들 사이의 0의 개수를 복호하게 된다. 이때 룩업 테이블은 zero_left와 입력 비트를 사용하여 복호한다. 여기서 zero_left는 현재 복호되고 있는 위치 이전의 0의 개수를 나타낸다. 이와 같이 zero_left 를 사용하여 복호할 경우 모든 계수사이를 복호할 필요 없이 zero_left가 0이 되는 지점에서 run_before의 복호가 끝나게 된다.

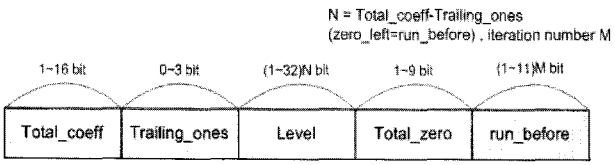

그림 2. CAVLC 복호화 비트 스트림
Fig. 2. Bitstream of CAVLC Decoding.

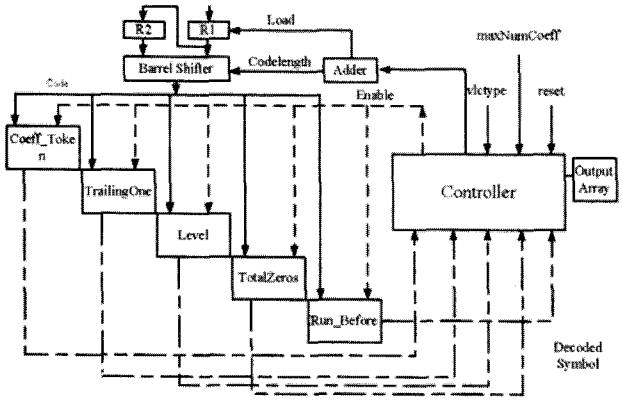

그림 3. CAVLC 복호화기 하드웨어 구조
Fig. 3. CAVLC Decoder Hardware Architecture.

위 5단계를 모두 복호한 후 복호된 값들을 조합하여 각 계수는 역방향 zigzag 스캐닝 순서대로 블록에 배치되어 레지듀얼 블록을 구성하게 된다. 각 단계별 복호화에 필요한 입력비트들은 그림 2와 같이 구성된 비트스트림의 순서로 각 단계에 입력비트로 사용된다. 비트스트림의 구성순서는 복호화 단계의 순서와 같다.[6]

2.2. 기존의 CAVLC 알고리즘을 적용한 구조

H.264/AVC의 CAVLC의 복호화 방식은 룩업테이블에 의존한다. CAVLC를 효율적으로 복호하기 위해 기존의 제안된 하드웨어 구조는 룩업테이블을 효율적으로 복호하는 방식인 Chang[2], Lee[6]과 zero_skip을 이용하여 계수가 존재하지 않을 경우 나머지 블록을 수행하지 않고 복호하는 방식인 Yu[3]이 있다. Chang[2]는 복호화를 빠르게 수행하도록 하기 위해서 레벨에 대한 복호화 블록을 룩업 테이블로 정의했다. 이 경우 복호과정에서 여러 번 반복되는 레벨의 복호를 룩업 테이블로 정의함으로써 복호화 수행시간을 줄일 수 있지만 룩업 테이블로 인해 하드웨어의 복잡도가 상당히 증가하게 된다. Lee[6]는 하드웨어 복잡도를 줄이기 위하여 주변 매크로블록에 대한 정보를 저장하는 메모리를 재사용하여 저장하는 구조와 total_coeff와 trailing_ones에 대한 테이블을 Prefix와 Suffix로 나누어 룩업 테이블을 구현하는 방법을 제안하였다. 그러나 이 경우에는 Prefix를 먼저 계산한 뒤 Suffix와 룩업 테이블을 참조해서 복호하기 때문에 하드웨어 복잡도는 줄어들지만 total_coeff와 trailing_ones를 복호하는데 4사이클이 걸리게 된다. Yu[3]는 레벨을 룩업 테이블이 아닌 prefix와 suffix_length를 이용하여 복호하는 방식을 사용하는 대신 수행시간을 줄이기 위해 total_coeff, trailing_ones, sign of trailing_ones을 한 사이클 안에 복호하는 방법과 total_coeff가 0인 경우 다음의 복호화 블록들을 수행하지 않고 복호할 수 있도록 설계되었다. total_coeff, trailing_ones, sign of trailing_ones을 한 사이클 안에 복호하기위해 total_coff 와 trailing_ones을 룩업 테이블을 이용하여 먼저 복호한 후 조합회로를 사용하여 trailing_ones값에 따라 sign of trailing_ones을 복호하도록 설계되어있다.

제시된 기존의 알고리즘들은 CAVLC 수행시간을 줄이기 위해 각 블록의 룩업 테이블을 효율적으로 복호하는 방법에 초점을 두었다. 그러나 각 블록간의 복호를 수행하는데 필요한 유효비트를 구하기 위해 소요되는 수행시간이 전체 CAVLC 수행시간에 미치는 영향에 대

한 고려는 하지 않았다. 따라서 기존의 알고리즘들은 모두 유효비트를 구하기 위하여 동일한 구조의 제안하고 있다. 기존의 알고리즘들이 적용한 유효비트를 구하기 위한 CAVLC 복호화기의 하드웨어 구조는 위에서 언급한 5단계의 복호단계에 따라 그림 3과 같다. 하드웨어의 전체적인 구조를 보면, 각 5단계를 복호화 하는 블록과 이를 컨트롤하는 블록 그리고 유효비트를 각 블록으로 전달하기 위한 배럴쉬프터와 Accumulator로 구성되어 있다. 각 블록별 기능은 다음과 같다.

CAVLC 알고리즘에 따라 먼저 Coeff_taken 블록의 total_coeff와 trailing_ones가 복호 된다. 이때 배럴쉬프터로부터 넘겨받는 입력 비트의 길이는 테이블의 가장 긴 코드워드인 16비트가 된다. 테이블을 참조하여 복호한 후 출력으로 total_coeff와 trailing_ones, 그리고 실제 복호된 코드워드의 길이를 출력한다. 여기서 코드워드의 길이는 다음 블록이 복호화를 하는데 필요한 유효비트를 구하는데 이용된다.

다음 두 번째 블록인 Trailing_one 블록에서는 Coeff_taken 블록에서 복호된 trailing_onse의 값에 따라 복호화가 진행된다. trailing_ones의 최대값은 3이기 최대코드 워드의 길이인 3비트를 입력으로 받게 된다. 이 블록에서는 복호된 Trailing_ones의 부호와 복호된 코드워드의 길이를 출력한다.

다음 레벨블록은 trailing_ones을 뺀 나머지 계수를 복호하기 때문에 (total_coeff-trailing_ones)개수 만큼 반복해서 복호를 진행한다. 이때 레벨의 코드워드의 길이는 최대 32비트이므로 32비트의 입력을 비트스트림으로 받아서 레벨 값을 복호하게 된다. 레벨 블록에서는 레벨 값과 복호된 코드워드의 길이를 출력한다.

Total_zero 블록은 최대 코드워드의 길이가 9비트이다. 따라서 total_zero는 total_coeff, 그리고 9비트의 입력비트스트림과 룩업 테이블을 참조하여 복호한다. 이 블록의 출력은 total_zero와 코드워드의 길이이다.

CAVLC 복호화의 마지막 5번째 단계인 run_before는 0이 아닌 계수 사이의 0의 개수를 복호하는 것이기 때문에 zero_left가 0이 아니면 total_coeff의 값만큼 복호를 반복한다. 이 블록은 룩업테이블을 참조하여 run_befor와 코드워드의 길이를 출력한다.

컨트롤러는 mb_type, slice_type, maxNumCoeff 등의 파라미터를 입력으로 받아 각 단계를 복호하는데 필요한 파라미터를 생성한다. 5가지의 단계의 블록을 순차적으로 Enable시키면서 적절한 파라미터를 넘겨주고 각 블록의 복호된 값과 코드워드의 길이를 넘겨받는다. 이때 블록으로부터 복호된 값을 이용하여 역방향 zigzag 스캐닝 순서대로 계수를 복호하여 4x4 또는 2x2 레지듀얼 블록을 구성하게 된다. 그리고 각 단계의 블록으로부터 받은 코드워드는 이전의 코드워드 길이와 더하는 과정을 거친 후, Accumulator에 계산된 값을 전달하여 다음 단계를 복호하는데 필요한 유효비트를 얻을 수 있도록 한다. Accumulator는 컨트롤러로부터 값을 받아 그 값을 threshold값과 비교판단 후, 배럴쉬프터를 shift-left하거나 한 번의 뺄셈의 과정 후 shift-left 함으로써 배럴쉬프터가 다음단계에 유효한 비트를 전달할 수 있도록 한다. 여기서 threshold값은 입력비트스트림의 길이와 같다.

이와 같이 유효비트를 구하기 위해서는 컨트롤러, 배럴쉬프터, Accumulator등 3단계의 과정과 그 외의 덧셈과 뺄셈 과정에 대한 추가적인 연산을 필요로 한다. 이는 CAVLC가 HD급 영상을 고속으로 처리하는데 문제가 된다. 앞서 말한 기존의 방식 또한 유효비트를 구하기 위해서 위와 같은 과정을 반복한다. 따라서 다음과 같은 문제를 야기한다. 첫 번째, 룩업테이블을 효율적으로 복호하는 방식은 룩업테이블을 효율적으로 복호하더라도 다음 복호화 블록에서 필요한 유효비트를 구하기 위해서는 여전히 컨트롤러와 Accumulator를 이용해야 하기 때문에 5단계를 모두 복호할 경우 블록간의 유효비트를 구하기 위한 수행시간이 늘어나게 되고 비효율적으로 동작하도록 만든다. 두 번째, zero_skip을 적용한 방식은 '0'이 아닌 계수가 적을 경우 나머지 블록을 skip할 수 있기 때문에 Chang[2] 알고리즘에 비해 효율적으로 동작하지만 '0'이 아닌 레지듀얼의 계수가 많을 때는 그렇지 못하다. 여기서, 레지듀얼 계수는 QP값에 의존하는데, QP값이 작을수록 레지듀얼의 계수가 많이 존재한다. 만일 QP가 작을 경우, zero_skip방식은 5단계의 블록을 모두 수행하게 되어 수행시간이 크게 증가한다. 이러한 문제를 해결하기 위해서는 크기가 매크로블록 혹은 프레임 단위의 코드워드 길이를 가지는 배럴쉬프터를 사용함으로써 위의 과정이 제거할 수 있으나, 영상의 특성에 따라 코드워드의 길이가 가변적으로 달라지기 때문에 하드웨어로 효율적인 구현을 하는 것은 적합하지 않다. 따라서 이러한 문제를 해결하기 위한 새로운 구조가 필요하다.

Ⅲ. 제안된 알고리즘을 적용한 CAVLC 구조

블록간의 유효비트를 구하는 횟수가 복호화 효율을

떨어뜨리는 문제를 해결하기 위해 본 논문에서는 두 가지 알고리즘을 제안한다. 하나는 total_Coeff블록과 trailing_ones블록을 하나의 블록으로 설계하여 처리함으로써 유효비트를 처리하는 단계를 기존의 5단계에서 4단계로 줄이는 것이고 다른 하나는 입력 비트스트림을 저장하는 64비트의 쉬프트 레지스터를 기존하드웨어에 추가하여 컨트롤러를 거치지 않고 쉬프트 레지스터를 shift-left함으로써 다음 블록에서 요구되는 유효비트를 빠르게 얻는 방법이다. 또한, 64비트 쉬프트 레지스터를 이용하기 위해 기존의 32비트 배럴쉬프터를 64비트로 확장하고, 입력버퍼 또한 32비트로 확장한다. 제안된 알고리즘을 적용할 경우 유효비트를 구하는 과정을 4단계로 줄일 수 있고, 유효비트를 구하기 위한 수행시간을 1사이클 줄임으로써 CAVLC의 전체 수행시간을 줄일 수 있다. 그림 4는 제안된 알고리즘을 적용한 CAVLC 하드웨어 구조이다. 제안된 구조에서는 기존의 두 개의 블록으로 구성되었던 Total_coeff 블록과 Trailing_ones의 블록을 하나의 블록으로 처리하여 복호화 단계를 5단계에서 4단계로 줄였다. 또한, 유효비트를 빠르게 구하기 위해서 64비트 쉬프트 레지스터와 각 블록의 코드워드의 길이를 적절히 선택할 MUX가 추가되었다.

3.1. Total_coeff&Trailing_ones 블록

Lee[6]에서는 테이블의 사이즈를 줄이기 위하여 코드워드를 첫 번째 1이 나오기 이전의 0개 개수를 Prefix, 뒤의 비트스트림을 Suffix로 나누어 룩업 테이블을 구현하였으나 기존보다 많은 사이클 수가 요구된다. 따라서 본 논문에서는 하드웨어 복잡도를 줄이기 위해서 total_coeff 블록과 sign of trailing_ones 블록을 하나의 블록으로 합치는 것을 제안한다. Yu[3]에서는 total_coeff 블록과 sign of trailing_ones 블록을 하나의 블록으로 설계하기 위해 기존의 룩업 테이블에 sign of trailing_ones을 복호하기 위한 조합회로를 추가적으로 설계하였으나, 본 논문에서는 하드웨어의 사이즈를 줄이기 위하여 별도의 조합회로를 사용하지 않고 total_coeff, trailing_ones, sign of trailing_ones를 한 사이클에 복호할 수 있는 룩업 테이블을 제안하였다.

total_coeff는 최대 16비트의 코드워드를 가지기 때문에 입력 비트스트림으로 16비트를 받는다. total_coeff의 코드워드 다음에 trailing_ones의 부호비트들이 연속적으로 나오기 때문에 total_coeff의 룩업 테이블을 만들 때 3비트를 추가함으로써 total_coeff와 trailing_ones의 값을 동시에 복호할 수 있다. 이 경우 total_coeff를 복호한 후 다음 trailing_ones의 부호비트를 복호하는데 필요한 유효비트를 얻기 위해 복호된 코드워드의 길이가 컨트롤러와 accumulator를 거치는 단계를 생략할 수 있다.

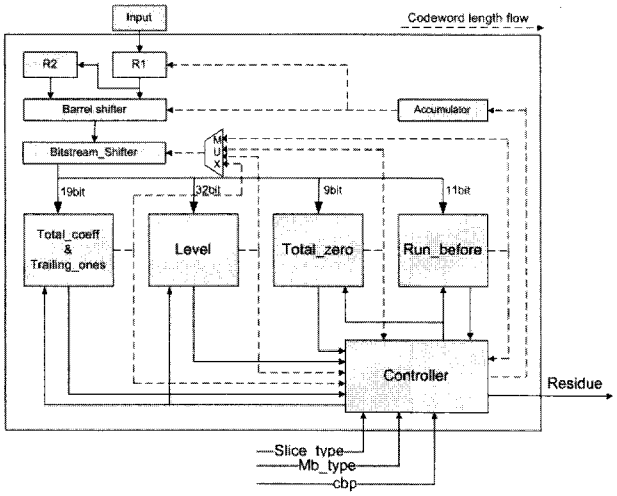

그림 4. 제안된 CAVLC 복호화기 하드웨어 구조
Fig. 4. Proposed CAVLC Decoder Hardware Architecture.

표 1. 0<=nC<2, Totall_coeff&Trailing_onse의 룩업 테이블
Table 1. Proposed lookup table, 0<=nC<2.

Trailing_ones	Total_Coeff	sign	0<=nC<2
0	0		1
0	1		0001 01
1	1	x′	01x
0	2		0000 0111
1	2	x′	0001 00x
2	2	x′x′	001x x
0	3		0000 0011 1
1	3	x′	0000 0110 x
2	3	x′x′	0000 101x x
∘	∘	∘	∘
∘	∘	∘	∘
∘	∘	∘	∘

다음 표 1은 total_coeff와 sign of trailing_ones 두 블록을 하나의 블록으로 구하기 위해 제안한 룩업 테이블의 예이다. trailing_ones의 부호는 total_coeff과 trailing_ones를 복호할 때, trailing_ones의 값에 따라 복호된다. trailing_ones의 값이 '1'이면 total_coeff를 복호한 마지막 비트 다음의 1비트를 이용하여 복호하고, '2'이면 2비트를 '3'이면 3비트를 이용하여 복호한다. 복호화 방법은 기존의 알고리즘과 동일하게 비트스트림의 x가 '0'이면 x'는 '+1'로 '1'이면 '-1'로 복호된다. 그러나 trailing_ones의 값이 '0'이면 복호화는 진행되지 않고 코드워드의 길이 또한 변하지 않는다.

따라서 total_coeff&trailing_ones를 복호하기 위해서 필요한 코드워드의 최대길이는 sign of trailing_ones의 최대 코드워드 길이 3비트가 더해진 19비트가 되고, 입력 비트스트림으로 19비트를 받는다. 따라서 다음 블록으로 넘겨주게 되는 코드워드의 길이는 total_coeff를 복호하는데 필요한 코드워드의 길이와 trailing_ones의 부호를 복호하는데 필요한 코드워드의 길이를 더한 값이 된다. 위와 같이 두 블록을 하나의 블록으로 처리함으로써 두 단계사이의 유효비트를 구하는 과정이 생략됨과 동시에 복호화의 두 단계를 동시에 복호할 수 있는 이점을 가진다. 이로써 코드워드가 컨트롤러와 Accumulator를 거쳐 유효비트를 구하는 단계를 생략할 수 있고, 두 단계를 거쳐 복호해야 하는 것을 하나의 테이블로 구현함으로써 한 사이클 안에 두 가지 심볼을 복호한다. 따라서 total_coeff와 trailing_ones를 더 효율적으로 복호할 수 있고, 전체 CAVLC 수행시간을 약 5% 줄일 수 있다.

3.2. 유효비트를 구하기 위한 딜레이를 줄이는 방법

제안된 알고리즘을 적용하여 total_coeff 블록과 trailing_ones 블록을 하나의 블록으로 처리함으로써 5단계의 복호화 과정 중 trailing_ones 부호를 복호하는데 필요한 유효비트를 구하는 절차를 생략할 수 있다. 그러나 레벨 블록과 run_before 블록은 한 번에 복호가 끝나지 않고 total_coeff의 수에 따라 복호화를 반복수행하기 때문에 여러 번의 유효비트를 구하는 것이 필요하다. CAVLC 알고리즘을 고속으로 처리하기 위해서는 복호 수행시간이 가장 긴 레벨 블록과 run_before 블록의 고속처리가 중요하다. 이 문제를 해결하기 위해 각 단계사이의 유효비트를 얻는 데 걸리는 시간을 줄이는 알고리즘을 제안한다.

그림 5는 제안된 알고리즘을 적용한 구조를 보여 준다. 전체적인 입력비트 스트림을 가진 64비트의 쉬프터 레지스터가 있고, 이 레지스터로부터 각 블록은 입력을 받게 된다. 기존의 알고리즘에서는 64비트의 쉬프터 레지스터 대신 32비트의 배럴쉬프터로부터 블록별로 9비트, 3비트, 32비트, 11비트, 9비트의 입력만을 받아 복호를 수행하고 코드워드의 길이와 심볼 값을 컨트롤러에 넘겨주었다. 그러나 제안된 알고리즘에서는 기존의 32비트 배럴쉬프터를 64비트 배럴쉬프터로 대체하고 64비트 쉬프트 레지스터와 MUX를 추가적으로 구현한다. 입력비트스트림은 64비트 배럴쉬프터를 통해 64비트 쉬프트 레지스터에 전달되고 64비트 쉬프트 레지스터는 전달된 입력비트스트림을 각 블록에 전달한다. 이때, 64비트 배럴쉬프터로부터 64비트 쉬프트 레지스터로의 입력비트스트림 전달은 코드워드의 길이가 64를 초과하여 새로운 입력비트스트림이 요구될 경우에만 이루어진다. 64비트 쉬프트 레지스터로부터 유효한 비트스트림을 전달 받은 각 복호 블록은 전달된 입력 비트스트림을 이용해서 복호화를 수행한 후, 코드워드의 길이를 컨트롤러에만 넘겨주지 않고 MUX를 통해 직접 64비트 쉬프트 레지스터를 shift-left 함으로써 유효비트를 구한다. 따라서 컨트롤러에만 코드워드의 길이를 넘겨줘서 Accumulator를 통해서만 다음 블록의 유효비트를 구하는 이전의 알고리즘에 비해 더 효율적으로 유효비트를 구할 수 있다.

제안된 알고리즘의 복호화 절차는 그림 6과 같다. 각 블록들은 입력을 받아서 복호화를 진행하게 된다. 복호화가 끝난 후 각 블록은 심볼 값과 코드워드의 길이를 출력하게 된다. 이때 복호 값은 컨트롤러에 의해 4x4레지듀얼 블록을 구성하는데 사용되고, 코드워드의 길이는 다음 유효비트를 구하기 위해 사용된다. 코드워드의

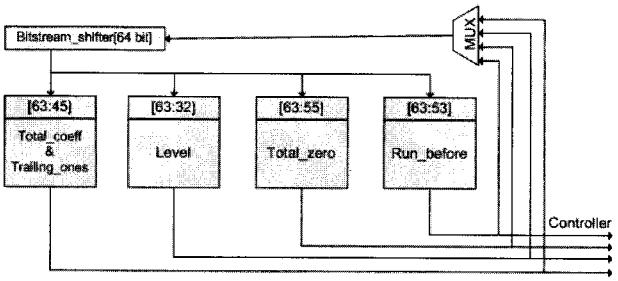

그림 5. 제안된 알고리즘 구조
Fig. 5. The architecture of proposed algorithm.

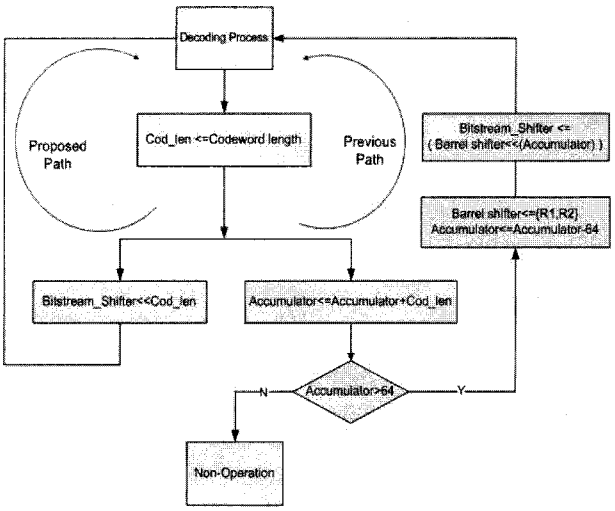

그림 6. 유효비트를 구하기 위한 flow chart
Fig. 6. Flow chart calculating efficient bits.

길이는 64비트의 쉬프트 레지스터를 shift-left하게 되고 쉬프트 되어진 64비트의 쉬프트 레지스터의 값은 다음 블록에 입력 비트스트림을 전달하게 된다. 이러한 과정은 total_coeff, trailing_ones, level, total_zero, run_before 등의 복호화 순서에 따라 복호화 절차가 끝날 때까지 반복하게 된다. 그러나 만일 복호할 레벨이나 run_before 값이 많을 경우 요구된 유효비트가 64비트를 초과하게 된다. 이 경우에는 기존의 알고리즘과 같이 컨트롤러에서 누적된 코드워드의 길이를 계산하여 Accumulator에 적절한 값을 저장한 후, 배럴쉬프터를 쉬프트 시킴으로써 복호하는데 필요한 유효비트를 구한다. 그림의 왼쪽의 Path는 제안된 알고리즘을 적용하여 빠르게 유효비트를 구하는 과정을 보여주며, 오른쪽의 Path는 누적된 코드워드의 길이가 64비트를 초과할 경우 기존의 알고리즘과 같은 방식을 사용하여 유효비트를 구하는 과정을 보여준다. 하지만 대부분의 영상에서 64비트 이내에 4화소x4화소 값들이 복호되기 때문에 3단계에 걸쳐 유효비트를 구하는 것보다 효율적이다.

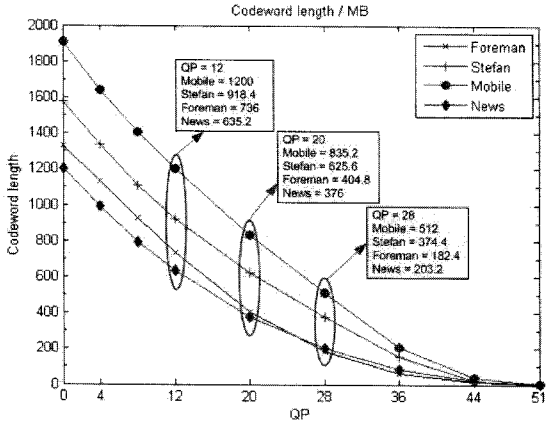

그림 7. 매크로블록 당 코드워드 길이
Fig. 7. codeword length per macroblock.

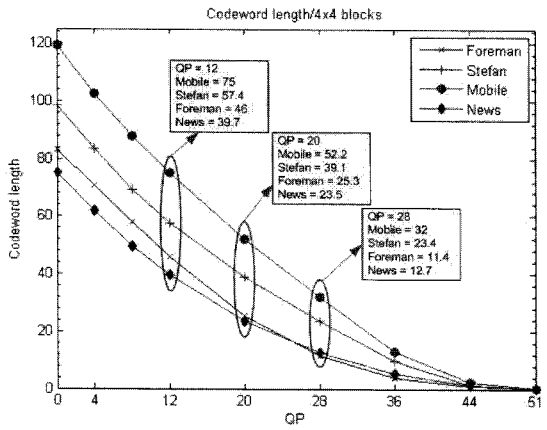

그림 8. 4x4블록 당 코드워드의 길이
Fig. 8. codeword length per 4x4 block.

다음 그림 7과 그림 8은 Mobile, Foreman, Stefan, News 영상에 대한 매크로블록 당 요구되는 코드워드의 평균길이와 4x4블록 당 요구되는 코드워드의 평균길이를 나타낸다. 4x4블록 당 요구되는 길이를 보면 QP=28, QP=20 일때, 모든 영상이 64비트 이내의 코드워드의 길이를 가짐을 알 수 있다. 따라서 기존 32비트 배럴쉬프터를 사용하는 것보다 64비트 단위의 배럴쉬프터와 쉬프터 레지스터를 사용함이 더 효율적임을 알 수 있다. 또한 QP=12 일때, Mobile 영상을 제외한 모든 영상이 64비트 이내의 코드워드의 길이를 가짐을 알 수 있다. 따라서 기존의 32비트 배럴쉬프터를 이용하여 복호할 때보다 새로운 비트스트림을 처리하는 과정을 줄임으로써 더 효율적으로 CAVLC를 복호할 수 있고, HD급 영상의 실시간 처리를 할 수 있다.

Ⅳ. 기존 알고리즘과의 분석

제안된 CAVLC구조의 성능을 평가하기 위해 기존의 CAVLC 알고리즘으로 레지듀얼계수의 복호화를 했을 때와 제안한 알고리즘을 이용하여 복호하였을 때의 매크로블록 당 처리속도를 비교하였다. 처리속도를 비교하기 위해 Mobile, Foreman, Stefan, News등 4가지 영상을 사용하였다. 4가지 영상에 대해서 QP 값에 따른 사이클 수가 어떻게 변하는지 알아보기 위해 QP 값을 12, 20, 28로 변화시켰으며, 동일한 라이브러리를 이용하여 합성되었으며, 기타 평균 사이클 수에 영향을 줄 수 있는 다른 모든 조건을 동일하게 설정하였다. 각 영상에 대한 실험은 Verilog HDL을 이용하여 시뮬레이션을 수행하였다.

표 2는 기존 알고리즘을 적용하였을 때와 제안된 알고리즘을 적용하였을 때의 매크로블록 당 사이클 수를 보여준다[3]. 비교 대상 중 가장 좋은 성능을 가진 Chang[2]와 Yu[3]에 대해서는 영상과 QP에 대해서 성능을 비교하였고, Lee[6]에 대해서는 Worst Case에 대해서만 비교하였다. Chang[2]와 Yu[3]에 대한 실험값은 Yu[3]을 참고하였다.

QP=28에 대해서 실험값은 Mobile 영상과 Stefan 영상의 경우 Yu[3] 대비 약 29~26%의 사이클 수가 줄었고, Foreman 영상과 News 영상의 경우 Yu[3] 대비 약 22~24%의 사이클 수가 줄어들었다. QP=20에 대해서 실험값은 Mobile 영상과 Stefan 영상의 경우 Yu[3] 대비 약 29~26%의 사이클 수가 줄었고, Foreman 영상과

표 2. 매크로블록 당 평균 사이클 수 (Cycle/MB)
Table 2. The average cycles per macroblock. (Cycle/MB)

QP	Sequence	Chang[2]	Yu[3]	Proposed	notice
28	Mobile	395	174	122	(29%)
	Foreman	192	53	41	(22%)
	Stefan	310	124	91	(26%)
	News	196	58	44	(24%)
20	Mobile	570	279	185	(33%)
	Foreman	306	120	90	(25%)
	Stefan	441	200	142	(29.5%)
	News	282	108	80	(25.9%)
12	Mobile	704	353	228	(35%)
	Foreman	468	214	155	(27.5%)
	Stefan	563	269	183	(32%)
	News	404	176	125	(28.9%)

표 3. 하드웨어 게이트 카운트
Table 3. Hardware cost analysis.

	Chang[2]	Yu[3]	Proposed
Library	0.18um	0.18um	0.18um
Gate-count	9.9K	13.1K	14.2K
RAM	11526bit		

News 영상의 경우 Yu[3] 대비 약 22~24%의 사이클 수가 줄어들었다. QP=12에 대해서 실험값은 Mobile 영상과 Stefan 영상의 경우 Yu[3] 대비 약 29~26%의 사이클 수가 줄었고, Foreman 영상과 News 영상의 경우 Yu[3] 대비 약 22~24%의 사이클 수가 줄어들었다. 제안한 복호화 알고리즘은 각 블록의 유효비트를 구하기 위해 컨트롤러와 Accumulator에서 의 불필요한 덧셈이나 뺄셈 과정을 생략하고, MUX를 통해 직접 64비트 쉬프트 레지스터를 shift-left함으로써 유효비트를 효율적으로 구하기 때문에 전체 수행시간을 줄일 수 있다. 이때, 레벨과 run_before의 반복 횟수가 많을수록 유효비트를 구하는 과정 또한 비례해서 증가하기 때문에 유효비트를 효율적으로 구하도록 제안한 구조가 더 효율적으로 동작하게 된다. 따라서 매크로블록 내에서 다른 영상에 비해 더 많은 계수를 가지는 Mobile 영상과 Stefan영상에서 효율적으로 동작함을 알 수 있고, QP 값이 작을수록 더 효율적으로 동작함을 알 수 있다.

Lee[6]는 Worst Case의 경우에 대해서만 비교하였다. Worst Case의 경우, Lee[6]은 total_coeff와 trailing_ones를 복호하기 위해 5사이클, 레벨을 복호하기 위해 30사이클, total_zero를 복호하기 위해 2사이클, run_before 및 레지듀얼 블록 구성을 위해 19사이클 등 4x4블록을 복호하기 위해 56사이클이 소요되며, 매크로블록을 복호하기 위해 1406사이클이 소요된다. 그러나 제안된 구조를 적용한 경우 total_coeff와 trailing_ones를 복호하기 위해 2사이클, 레벨을 복호하기 위해 30사이클, total_zero를 복호하기 위해 2사이클, run_before 및 레지듀얼 블록 구성을 위해 17사이클 등 4x4블록을 복호하기 위해 51사이클이 소요되며, 매크로블록을 복호하기 위해 1258사이클이 소요된다. 따라서 10.5% 정도의 사이클 수가 줄었다.

표 3은 기존 알고리즘을 적용한 복호화기와 제안된 알고리즘을 적용한 복호화기의 하드웨어 코스트를 비교한 것이다. 제안된 하드웨어 구조는 Verilog HDL을 이용하여 설계 및 검증되었고, 0.18um 표준 셀 라이브러리를 사용하여 합성하였다. Chang[2]의 게이트 카운트는 9.9K이고, Yu[3]는 13.1K, Lee[6]은 11.7K이고 제안된 알고리즘은 14.2K이다. 그러나 Chang[2]과 Lee[6]의 경우 RAM (11526bit)과 RAM (7312bit)을 사용하였으나 Yu[3]와 제안된 구조는 RAM을 사용하지 않는다.

V. 결 론

본 논문에서는 효율적으로 레지듀얼 계수를 복호하는 하드웨어 설계 구조를 제안하였다. 기존의 알고리즘은 유효비트를 구하기 위해서 컨트롤러 블록과 Accumulator 블록을 거쳐야했다. 이는 레벨이나 run_before 블록을 여러 번 반복해서 복호할 경우 유효비트를 구하는 횟수 또한 복호횟수에 비례해서 늘어나기 반복횟수가 많아질수록 CAVLC는 더욱 비효율적으로 동작하게 된다. 또한 Total_coeff 블록과 Trailing_ones 블록을 각각 수행함에 따라 복호화가 비효율적일 뿐 아니라 각 블록의 유효비트를 구하는 단계가 추가되어 사이클 수가 늘어나게 된다. 따라서 Total_coeff 블록과 Trailing_ones블록을 하나의 블록으로 설계하고 각 블록들이 요구하는 유효비트를 효율적으로 구함으로써 복호화 수행시간을 줄이는 구조를 제안하였다. 제안된 구조를 적용하여 실험한 결과, 하나의 매크로블록을 처리하는 평균 사이클 수가 기존의 제안된 구조보다 약 22~29%가 줄어들었다. 따라서 기존의 제안된 알고리즘에 비해 더 낮은 클럭에서 HD급 영상 1080i를 실시간으로 처리할 수 있다.

참 고 문 헌

[1] J. V. Team, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification," ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, pp. 148-159, May 2003.

[2] Hsiu-Cheng Chang, Chien-Chang Lin, Jiun-In Guo, "A novel low-cost high-performance VLSI architecture for MPEG-4 AVC/H.264 CAVLC decoding", International Symposium on Circuits and Systems 2005, pp. 6110-6113, May 2005.

[3] Guo-Shiuan Yu, Tian-Sheuan Chang, "A zero-skipping multi-symbol CAVLC decoder for MPEG-4 AVC/H.264", International Symposium on Circuits and Systems 2006, pp. 5583-5586, May 2006.

[4] Esra Sahin, Ilker Hamzaoglu, "A high performance and low power hardware Architecture for H.264 CAVLC Algorithm", 13th European Signal Process Conference, September 2005.

[5] Heng-Yao Lin, Ying-Hong Lu, Bin-Da Liu, Jar-Ferr Yang, "Low power design of H.264 CAVLC decoder", International Symposium on Circuits and Systems 2006, pp. 2689-2692, May 2006.

[6] Dae-joon Lee, Yong-jin Jeong, "VLSI architecture design of CAVLC entropy encoder/decoder for H.264", 한국통신학회 논문지 제30권 5C호, pp. 371~381, May 2005

저 자 소 개

오 명 석(학생회원)
2006년 경희대학교 전자공학과 학사 졸업.
2007년~현재 연세대 전기전자 공학과 석사 과정.
<주관심분야 : 영상처리, SoC 설계>

김 재 석(정회원)
1977년 연세대 전자공학과 학사 졸업.
1979년 KAIST 전기전자공학과 석사 졸업.
1988년 Rensselaer Polytechnic Institute, NY, 박사 졸업.
1993년~1995년 한국전자통신연구원 책임연구원
1996년~현재 연세대학교 전기전자공학과 교수
<주관심분야 : 통신 및 영상 시스템, VLSI 신호처리, 임베디드 S/W 및 SoC 구현>

이 원 재(학생회원)
2001년 연세대 전기전자공학과 학사 졸업.
2003년 연세대 전기전자공학과 석사 졸업.
2003년~현재 연세대 전기전자 공학과 박사 과정.
<주관심분야 : 영상처리, SoC 설계>