2013년 5월 전자공학회 논문지 제 50 권 제 5 호
Journal of The Institute of Electronics Engineers of Korea Vol. 50, NO. 5, May 2013 http://dx.doi.org/10.5573/ieek.2013.50.5.112

논문 2012-50-5-13

UD(Ultra Definition) 동영상 실시간 처리를 위한 H.264/AVC CAVLC 병렬 아키텍처 설계

(Parallel Architecture Design of H.264/AVC CAVLC for UD Video Realtime Processing)

고 병 수*, 공 진 흥**

(Byung Soo Ko© and Jin-Hyeung Kong)

요 약

본 연구에서는 UHD(3840×2160)영상을 실시간 처리하는 고성능 H.264/AVC CAVLC 부호화기를 설계하였다. 연산처리 성능을 높이기 위해 통계값 탐색 과정과 코드워드 부호화 과정을 각각 1사이클에 처리하도록 설계하였다. 통계값 탐색과정을 1사이클에 처리하기 위해 16개 계수들의 '0' 또는 '0'이 아님을 표시하는 비트열을 만들어 산술 및 논리연산을 통해 통계값을 한 번에 구하였다. 그리고 코드워드 부호화 과정을 1사이클에 처리하기 위해 레벨의 코드워드 길이를 결정하는 계수들과 임계값들과의 비교 연산을 동시에 처리함으로써 코드워드 부호화 과정의 재귀적 연산을 제거하였다. 제안하는 H.264/AVC 병렬 CAVLC 부호화기는 통계값 탐색 단계과 코드워드 부호화 단계로 나뉘는 2단 파이프라인 구조로 고속 병렬 연산 회로를 구현하였으며, 산술 연산을 적용하여 코드워드 부호화 테이블을 회로의 크기를 줄이고자 하였다. 0.13um 공정에서 시뮬레이션한 결과, 게이트 수는 33.4Kgates이며, 최대동작주파수 100MHz에서 UD 영상을 초당 100프레임으로 실시간 처리가 가능하다.

Abstract

In this paper, we propose high-performance H.264/AVC CAVLC encoder for UD video real time processing. Statistical values are obtained in one cycle through the parallel arithmetic and logical operations, using non-zero bit stream which represents zero coefficient or non-zero coefficient. To encode codeword per one cycle, we remove recursive operation in level encoding through parallel comparison for coefficient and escape value. In oder to implement high-speed circuit, proposed CAVLC encoder is designed in two-stage {statical scan, codeword encoding} pipeline. Reducing the encoding table, the arithmetic unit is used to encode non-coefficient and to calculate the codeword. The proposed architecture was simulated in 0.13um standard cell library. The gate count is 33.4Kgates. The architecture can support Ultra Definition Video (3840×2160) at 100 frames per second by running at 100MHz.

Keywords : Ultra Definiction, H.264/AVC, CAVLC, parallel architecture, Arithmetic Coding

Ⅰ. 서 론

여기에H.264/AVC는 가변블록의 움직임 보상 및 1/4 화소 단위의 움직임 추정, 화면 내 예측, 정수 기반 DCT와 블록현상 제거를 위한 블록현상 제거 필터, 그리고 비트단위 압축을 수행하는 엔트로피 부호화 등 다양한 압축 기술들은 사용하여 연산량이 크게 증가되었다[1]. 증가된 연산을 실시간 처리하기 위해 H.264/AVC의 다양한 압축 기술들에 대한 하드웨어 구조 설계가 진행 중이다. H.264/AVC CAVLC는 4x4블록 내 잔여화소들을 지그재그 스캔하여 통계값을 탐색하고 통계값을 코드워드로 부호화한다. 그런데 CAVLC의 처리과정이 순차적이기 때문에 처리성능이 낮아 병렬 고속처리하기

* 정회원, ** 평생회원, 광운대학교 컴퓨터공학과
(Department of Computer Engineering, Kwangwoon University)
※ "본 연구는 지식경제부가 지원하는 산업융합원천기술개발사업을 통해 개발된 결과임을 밝힙니다. (10039173 융복합 혁신반도체 기술개발)"
© Corresponding Author(E-mail: kfc78@daum.net)
접수일자: 2013년2월2일, 수정완료일: 2013년4월24일

위한 하드웨어 구조 설계가 다양하게 진행되고 있다[2~8].

CAVLC를 하드웨어로 구현한 대다수의 연구들은 통계값 탐색과정과 통계값 코드워드 부호화 과정을 병렬 고속화하고 있다. CAVLC를 하드웨어로 구현한 초기연구[2]는 통계값 탐색과정에서 4x4블록의 16개 계수를 스택 메모리에 넣었다가 하나씩 꺼내어 통계값을 구한다. 그리고 룩업테이블(LUT)이 아닌 산술 연산기로 코드워드를 부호화하여 메모리 크기를 줄였다. 그러나 스택 메모리에서 계수를 하나씩 꺼내어 통계값을 탐색하는 과정은 긴 처리시간을 요구하여 저해상도 영상만 실시간 처리하였다. 코드워드 부호화 과정을 병렬화시킨 연구[3]는 버퍼에 저장된 통계값들을 동시에 검색하여 코드워드로 부호화하였다. 그러나 통계값 탐색과정이 순차적으로 처리되어 통계값 탐색과정과 코드워드 부호화 과정의 처리성능이 불균형을 이루어 CAVLC 부호화기의 성능을 높이지는 못하였다. 통계값 탐색 과정과 코드워드 부호화 과정을 파이프라인으로 처리하는 연구[4]는 기존 연구[3]에 비해 처리성능을 25% 향상시켰다. 그러나 통계값을 순차적으로 코드워드 부호화하여 전체 처리성능 향상에는 한계가 있었다. 이후 연구들[5~6]은 통계값 탐색 과정과 코드워드 부호화 과정을 파이프라인으로 처리하며, 통계값 탐색 과정과 코드워드 부호화 과정을 고속처리하기 위한 방법들을 제안하였다. 계수 위치 탐색 기법과 레벨 순차 정렬 기법을 사용한 연구[5]는 기존 16개의 계수들을 모두 탐색하던 방법을 계수 위치탐색을 통해 탐색횟수를 줄여 통계값 탐색 과정을 고속화하고, 레벨 순차 정렬 기법을 통해 유효한 레벨과 T1(Trailing one)의 통계값을 선별하여 불필요한 코드워드 부호화 과정을 줄였다. 그리하여 기존연구[4]에 비해 처리성능이 20%정도 향상되었으나, 기존연구와 같이 Full HD (1920×1080)영상까지 실시간 처리하는 한계를 갖는다. 파이프라인의 단위를 기존 매크로 블록에서 블록단위로 줄인 연구[6]은 파이프라인의 지연시간(Latency)을 줄여 처리성능을 높이고, 버퍼의 크기도 작게 하여 회로의 크기를 줄였다. 또한 코드워드 부호화 과정을 병렬로 동시에 처리하여 코드워드 부호화 과정의 처리성능을 향상시켰다. 그리하여 기존연구[3~4]에 비해 처리성능이 향상되었으나, 여전히 Full HD 동영상을 실시간 처리하는 수준에 머물러 있다. 계수들의 지그재그 스캔과정을 순방향으로 탐색하는 연구[7]는 역방향 지그재그 스캔 과정을 없애 통계값 탐색 과정을 줄였으며, 레벨의 코드워드 부호화 과정을 제외한 나머지 통계값의 코드워드 부호화 과정을 한 사이클에 처리하도록 설계하여 전체처리성능을 높이고자 하였다. 그 결과 기존 연구들에 비해 2배의 처리성능을 보였으나, 레벨의 코드워드 부호화 과정에서 '0'이 아닌 계수들이 많을수록 처리 사이클이 증가하는 문제가 발생하여 초고화질 영상인 UHD(3840×2160) 동영상을 실시간 처리하지는 못하고 있다. 역방향 탐색과 순방향 탐색을 동시에 처리하는 연구[8]는 레벨을 역방향 탐색하고, 나머지 통계값을 순방향 탐색함으로써 통계값을 정렬하는 과정을 줄이고자 하였다. 그 결과 계수들을 순방향으로 통계값을 탐색하는 연구[7]에 비해 처리성능이 50%이상 향상되었으나, 여전히 레벨의 부호화 과정에서 '0'이 아닌 계수의 개수에 따라 처리 사이클이 늘어나 UHD (3840×2160) 동영상과 같은 초고화질 동영상을 실시간 처리하지 못한다. 따라서 기존 CAVLC 연구들은 고화질의 영상일수록 부호화해야 할 계수들의 개수가 늘어나 성능개선 효과가 적고, 양자화 레벨에 따른 가변적 제어로 인해 제어의 복잡도가 증가하는 문제를 가진다.

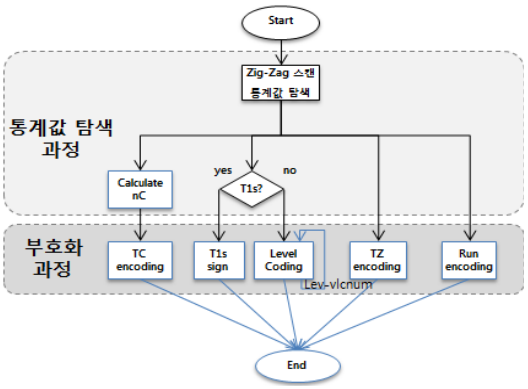
본 논문에서는 양자화 레벨과 관계없이 통계값 탐색과정과 코드워드 부호화 과정을 각각 1사이클에 처리하여 UHD 동영상을 실시간 처리하는 고성능 H.264/AVC 병렬 CAVLC를 제안하고자 한다. 먼저 16개 계수들이 '0'인지 아닌지를 표기하는 플래그(flag)형태의 16비트 열을 생성하여 산술 및 논리연산을 통해 1사이클만에 통계값을 구한다. 그리고 코드워드 길이를 결정하기 위한 레벨의 코드워드 부호화 과정에서 계수들과 임계값들과의 비교 연산이 재귀적으로 이루어지던 것을 동시에 비교 연산하도록 변경하여 레벨을 포함한 모든 코드워드 부호화 과정을 1사이클에 처리하였다. 제안하는 H.264/ AVC CAVLC는 통계값 탐색 단계와 코드워드 부호화 단계로 나뉘는 2단 파이프라인 구조로 설계하여 고속 병렬 연산 회로를 구현하였으며, 산술 연산을 적용하여 코드워드 부호화 테이블을 줄여 회로의 크기를 줄이고자 하였다.

논문의 구성은 다음과 같다. Ⅱ장에서는 H.264/AVC CAVLC 병렬 부호화 과정을 소개하고, Ⅲ장에서는 본 연구에서 제안한 H.264/AVC CAVLC 병렬 부호화기의 아키텍처를 설명하며, Ⅳ장에서는 실험 및 결과에 대하여 비교분석하고, Ⅴ장에서는 결론을 맺는다.

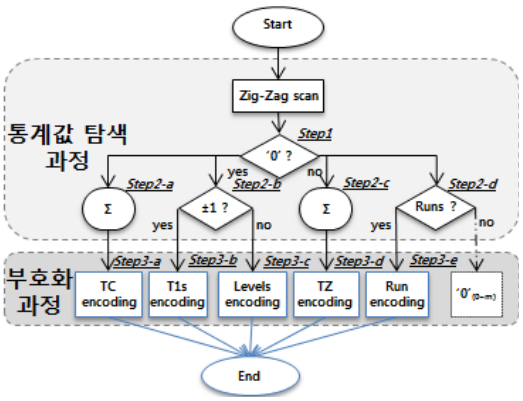
Ⅱ. H.264/AVC CAVLC 병렬 부호화

대다수의 CAVLC 연구들은 그림 1 (a)와 같이 4x4블

록의 계수들을 지그재그 스캔하여 통계값들을 탐색하고, 탐색된 통계값들을 코드워드로 부호화한다. 그림 1 (a)의 통계값 탐색 과정은 버퍼에 저장된 모든 계수들을 꺼내어 '0'이 아닌 계수들의 개수를 세고, 계수가 '±1'인지 여부를 확인하는 등의 과정을 순차적으로 처리하여 수행시간이 길다. 또한 그림 1 (a)의 코드워드 부호화 과정은 레벨의 코드워드 부호화 과정에서 계수들과 임계값들의 비교하여 코드워드 부호화 길이를 결정하는데, 이전 비교 연산의 결과에 따라 임계값들은 증가한다. 따라서 레벨의 코드워드 부호화 과정에는 계수들과 임계값간의 비교 연산이 재귀적으로 이루어져 전체 코드워드 부호화 과정의 수행시간이 길어진다. 따라서 제안하는 CAVLC는 그림 1 (b)와 같이 통계값 탐색 과정과 코드워드 부호화 과정을 병렬 연산 처리하여 각 과정의 수행시간을 1사이클로 줄이고자 하였다. 먼저 16개 계수의 값을 '0'인지 아닌지를 확인할 수 있는 플래그(Flag) 형태의 16비트 열인 "Non-zero flag"를 논리연산을 통해 구하고, "Non-zero flag"를 산술연산과 논리연산을 통해 '0'이 아닌 계수의 개수, '0'인 계수들의 개수 등의 통계값들을 1사이클에 계산한다. 그리고 레벨의 코드부호화 과정에서 계수들과 모든 경우의 임계값들을 동시에 비교하여 이전 비교 연산 결과에 상관없이 코드 부호화의 길이를 결정하도록 하여 재귀적인 연산을 제거하여 코드워드 부호화 과정의 병목인 레벨의 부호화과정을 1사이클에 처리하도록 설계하였다.



(a) 기존 CAVLC 처리과정



(b) 제안하는 CAVLC 처리과정

그림 1. CAVLC 부호화 과정
Fig. 1. CAVLC Encoding flow.

1. 통계값 탐색과정

통계값을 논리 및 산술연산으로 1사이클에 구하기 위해 먼저 "Step 1"에서 계수의 유무를 표시하는 Flag 형태의 16비트 열인 "Non-zero flag"를 생성한다. 그림 2와 같이 지그재그 스캔된 4x4블록의 16개 계수에 대해 각 계수를 OR 논리 연산하면 '0'이 아닌 계수들은 '1'로 표기되고, '0'인 계수들은 '0'으로 표기되는 "Non-zero flag"를 구할 수 있다. 그리고 "Step 2-a"에서 그림 3과 같이 "Non-zero flag"의 '1'을 합하면 '0'이 아닌 계수들의 총 개수를 구할 수 있다.

"Step 2-b"에서는 '0'이 아닌 계수를 연속하는 "±1"의 계수와 레벨으로 구분하여 통계값을 구한다. 먼저 최대 3개까지 가질 수 있는 연속하는 ±1의 계수를 찾기 위해 그림 4와 같이 "Non-zero flag"를 우선순위 디코더를 사용하여 '0'이 아닌 계수 3개를 검색하고 절대값을 확인하여 연속하는 ±1의 계수를 찾는다. 우선순위 디코더 기반의 T1 position detect는 "Non-zero flag"를 부호화 방향으로 '0'이 아닌 계수 3개를 검색한다. 그리고 3개 계수의 절대값을 확인하여 '1'일 경우에는 '1'로, '1'이 아

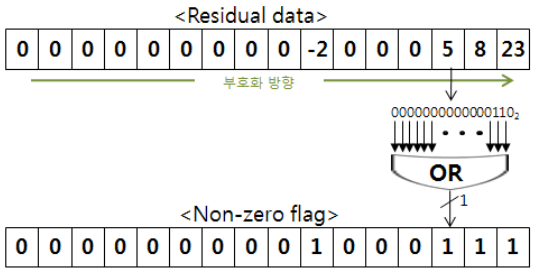


그림 2. Non-zero flag
Fig. 2. Non-zero flag.

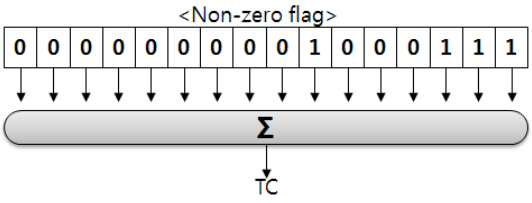


그림 3. Total Coefficient 추출 과정
Fig. 3. Extraction process of Total Coefficient.

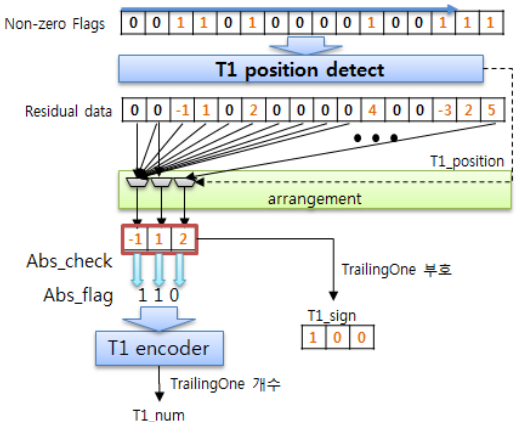


그림 4. Trailing Ones 추출 과정
Fig. 4. Extraction process of Trailing Ones.

표 1. Abs_flag를 이용한 TrailingOnes
Table 1. TrailingOnes for Abs_flag.

Abs	TrailingOnes
000	0
001	0
010	0
011	0
100	1
101	1
110	2
111	3

그림 5. Level flag 검색 과정
Fig. 5. Detect process of level.

닌 경우에는 '0'으로 표기하는 "Abs_flag"를 생성한다. "Abs_flag"가 연속하는 ±1의 계수 여부를 표기하므로 표 1을 이용하여 연속하는 ±1의 개수를 구한다. 그리고 3개 계수의 부호만 따로 부호화하여 3비트 형태의 연속하는 ±1의 계수의 부호를 얻는다.

연속하는 ±1의 계수를 제외한 나머지 계수들은 레벨이다. 따라서 그림 5와 같은 방법으로 "Non-zero flag"에서 연속하는 ±1의 계수를 뺀 나머지 레벨의 위치를 구한다. 마지막 ±1의 계수 위치 이후부터 '1'로 채워진 "Level Mask"를 생성하고, "Level Mask"를 "Non-zero flag"와 AND 논리 연산하여 "Level Flag"를 구한다. "Level Flag"는 코드워드 부호화 과정에서 '0'도 아니고 '±1'도 아닌 계수만을 선별할 수 있어 레벨의 코드워드 부호화 연산을 줄이는 데 사용된다.

"Step 2-c"에서는 마지막 '0'이 아닌 계수 앞에 존재

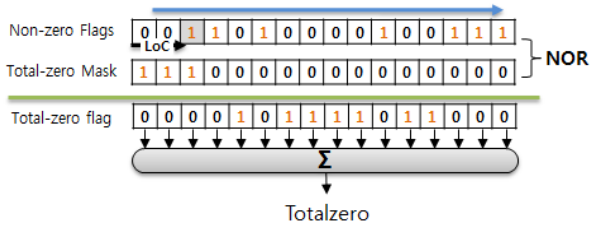
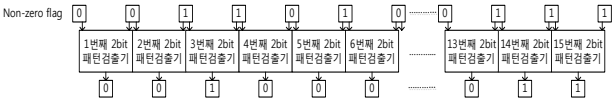


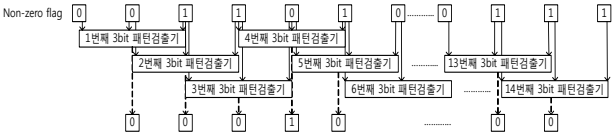
그림 6. Total-zero를 구하는 과정
Fig. 6. Total-zero computation process.

하는 '0'의 개수를 구한다. 먼저 "Non-zero flag"의 '0'이 아닌 계수의 마지막 위치(Last of Coefficient, LoC)를 탐색한다. 그림 6과 같이 '0'이 아닌 계수의 마지막 위치까지 '1'로 채워진 "Total-zero Mask"를 생성한다. 이렇게 얻어진 "Total-zero Mask"를 "Non-zero flag"와 NOR 논리 연산하면, 마지막 '0'이 아닌 계수 앞에 존재하는 '0'의 위치를 나타내는 "Total-zero flag"를 구할 수 있다. 그리고 "Total-zero flag"를 모두 합하면, 마지막 '0'이 아닌 계수 앞에 존재하는 '0'의 개수를 얻을 수 있다.

"Step 2-d"에서는 그림 7과 같이 "Non-zero flag"를 2~16bit 단위로 나누어 "11, 101, 1001, …, 1000 0000 0000 00001"의 비트열에 대하여 NOR 논리 연산기반의 패턴 검출하여 계수 사이의 연속된 '0'의 개수 및 위치 검색한다. 먼저 그림 7의 (a)와 같이 "Non-zero flag"를 2bit 단위로 나누어 2bit 패턴 검출기와 비교한다. 2비트 단위의 "Non-zero flag"값이 2bit 패턴 "11"과 같으면 2bit 패턴 검출기의 출력 값을 '1'로 표기하고 다르면 '0'으로 표기한다. 그리고 그림 7의 (b)에서와 같이 "Non-zero flag"를 3bit 단위로 나누어 3bit 패턴 검출기와 비교하여 3bit 패턴 "101"과 같으면 '1'로, 다르면 '0'으로 출력한다. 위와 같은 방법으로 "Non-zero flag"를 2~16bit의 패턴 검출하면 그림 8과 같이 패턴 검출의 출력 값을 얻을 수 있으며, 출력 값을 이용하여 시작



(a) 2bit 패턴 검출 과정



(b) 3bit 패턴 검출 과정

그림 7. Run Before 패턴 검색 과정
Fig. 7. Run Before pattern detect process.

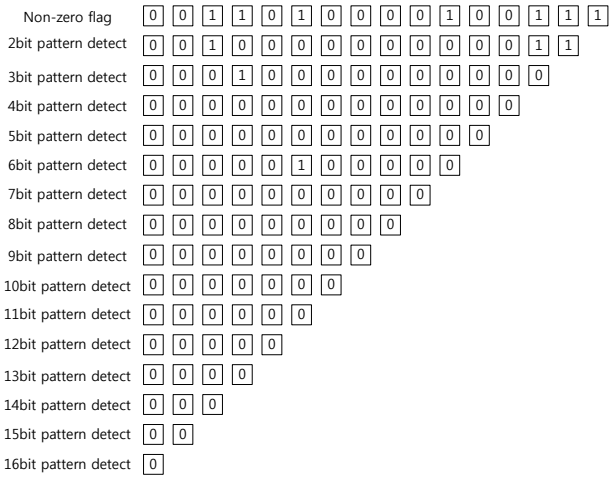


그림 8. 패턴 검출 출력 행렬
Fig. 8. Pattern detect output matrix.

하는 '1'과 끝나는 '1'사이에 연속하는 '0'의 개수와 위치를 구할 수 있다. 2bit 패턴 검출기의 n번째 출력 값이 '1'이면, n번째 계수 위치에서 n+1번째 계수 사이에는 연속하는 '0'이 없음을 나타내고, 3비트 패턴 검출기의 n번째 출력값이 '1'이면, n번째 계수와 n+2번째 계수 사이에는 연속하는 '0'이 1개 있음을 의미한다.

2. 코드워드 부호화 과정

코드워드 부호화 과정에서는 "Step 2"에서 얻어진 통계값들을 각각의 코드워드와 코드길이로 부호화한다. 먼저 "Step3-a"에서는 통계값 탐색과정에서 구한 '0'이 아닌 계수의 총 개수와 연속하는 ±1의 계수의 개수를 이용하여 하나의 코드워드로 부호화한다. "Step 3-b"에서는 통계값 탐색과정에서 구한 연속하는 '±1'의 계수의 개수와 부호를 하나의 코드워드와 코드길이로 부호화한다. "Step 3-c"에서는 '0'도 아니고 연속하는 '±1'도 아닌 변환계수들, 즉 레벨들을 코드워드와 코드길이로 부호화한다. 레벨의 코드부호화 과정은 그림 9와 같이 첫 번째 변환계수(Residual Data)의 절대 값을 초기 임계값(Escape Value)과 비교하는데, 첫 번째 변환계수가 임계값보다 작으므로 레벨 값은 0이다. 임계값은 표2와 같이 레벨 값(Lev VLC_num)에 의해 결정된다. 그리고 두 번째 변환계수(-1)와 임계값(0)과 비교하는데, 변환계수의 절대값이 임계값보다 크므로 세 번째 레벨 값이 '1'로 증가한다. 이와 같이 변환계수와 임계값의 비교 연산이 이전 변환계수와의 임계값 비교 연산에 결과에 따라 영향을 받는 데이터 종속성이 발생한다. 따라서 변환 계수와 임계값 비교 연산은 재귀적이고 반복적으로 이루어지며, 이로 인해 코드 부호화 과정의 수행시간이 길어지게 된다. 그런데 16개의 모든 계수들을 모든 임계값과 동시에 비교하면 레벨 값을 동시에 구할 수 있으며, 따라서 레벨의 코드 부호화 과정에서 레벨들을 코드워드와 코드길이로 동시에 부호화할 수 있다. 그림 10과 같이 모든 16개 계수들을 모든 임계값과 동시에 비교하면 재귀적으로 이뤄지던 데이터 종속성이 사라져 1사이클에 레벨을 코드워드와 코드길이로 부호화할 수 있다. "Step 3-d"에서는 계수 앞에 있는 모든 '0'의 개수에 따라 룩업테이블을 선택하여 부호화한다. "Step 3-e"에서는 통계값 탐색에서 구한 계수 사이의 연속된 '0'의 개수 및 위치를 가지고 룩업테이블을 선택하여 부호화한다.

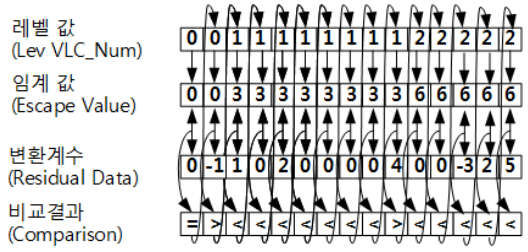


그림 9. 레벨 값 결정과정
Fig. 9. Decision process for Lev VLC_num.

표 2. 레벨 값에 따른 임계값
Table 2. Threshold Value by Lev VLC_num.

레벨 값	임계값
0	0
1	3
2	6
3	12
4	24
5	48
6	N/A

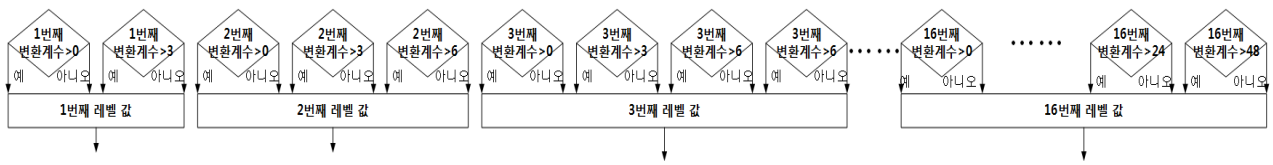


그림 10. 계수와 임계값 비교
Fig. 10. Comparison for coefficient and threshold value.

Ⅲ. 병렬 엔트로피 부호화 아키텍처

제안하는 H.264/AVC 병렬 CAVLC 부호화기는 그림 11에 나타난 바와 같이 통계값 탐색 (Scan)블록과 코드워드 부호화(Codeword Coding)블록으로 나뉜다. 통계값 탐색 블록은 4x4 block의 residual data를 zigzag 방향으로 정렬하는 Data Reorder블록, '0'과 '0'이 아닌 계수로 구분하기 위한 Non_zero Checker블록, '0'이 아닌 계수의 총 개수를 세는 Coeff. Adder블록, T1과 Level를 구분하는 T1 detector와 Level detector, 부호화될 '0'만을 구분하는 Zero detector, 계수사이의 '0'의 개수를 세는 Zero pattern detector로 구성된다. 스캔블록에서 구한 '0'이 아닌 계수의 총 개수 TC (Total_coefficient), 연속하는 T1의 개수(T1_num), T1의 부호(T1 Sign), Level이 있는 계수 위치를 나타내는 Level Flag, Zero Flag, Run Flag 및 Run Before 들은 코드워드 부호화 블록에서 입력되어 코드워드와 코드길이(codelength)로 부호화된다. 코드워드 부호화 블록은 5가지 부호화를 진행하는 각각의 부호화 블록(Coffe_Token encoding, T1_Sign encoding, Level encoding, Total Zero encoding, Run Before encoding)으로 구성되어 부호화 심볼을 코드워드와 코드길이(codelength)로 부호화한다.

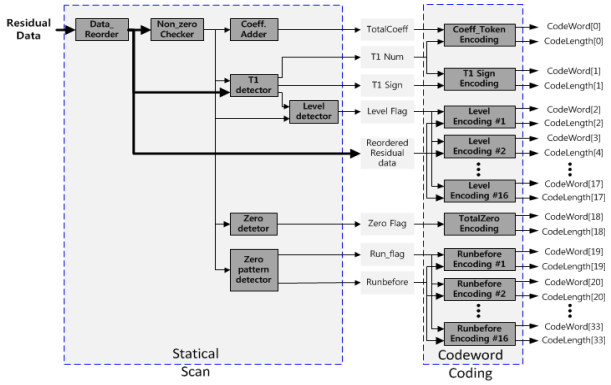


그림 11. 제안하는 CAVLC 아키텍처
Fig. 11. Proposed CAVLC architecture.

1. 스캔 블록

그림 12의 구조를 갖는 스캔블록은 먼저 Data Reorder 블록에서 4x4블록 데이터를 Zig-zag 스캔 순서로 재배열한다. 그리고 Non-zero flag 블록에서 4x4 블록 내 심볼을 OR 논리연산을 하여 계수의 유무를 표시하는 Non-zero flag를 생성한다. Coeff. Adder는 16 비트의 Non-zero flag를 모두 합하여 Total_Coeff(TC)를 구한다. T1 position detect는 우선순위 인코더 기반

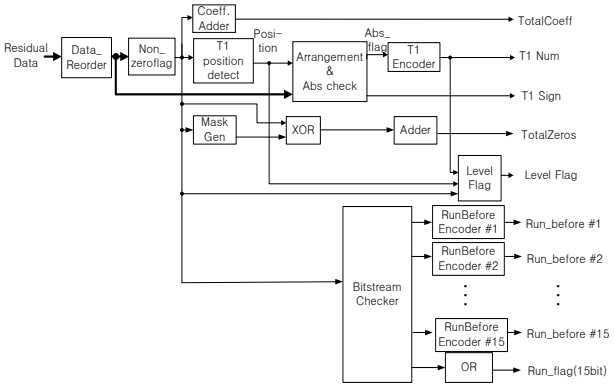


그림 12. 스캔블록의 구조
Fig. 12. Scan block architecture.

구조로 부호화 방향으로 '0'이 아닌 계수 3개를 검색한다. Arrangement & Abs check 블록에서는 검색된 계수들의 절대값을 확인하여 '1'일 경우에는 '1'로, '1'이 아닌 경우에는 '0'으로 표기하는 Abs_flag와 계수들의 부호를 나타내는 T1_Sign을 출력한다. T1 Encoder에서는 Abs_flag를 표 2를 활용하여 연속하는 T1의 개수(T1_num)를 구한다. 그리고 3개 계수의 부호만 따로 부호화하여 3비트 형태의 T1_sign를 얻는다. Total Zero의 연산은 먼저 Mask Gen 블록에서 첫 번째 계수 이후 1로 채워진 마스크를 생성한다. 그리고 마스크와 Non-zero flag를 XOR 논리연산을 하여 Total Zero의 위치를 찾고 CLA(Carry Lookahead Adder)기반의 덧셈기를 통하여 Total Zero를 구한다. T1 position detect가 찾아낸 T1의 위치와 T1 인코더의 T1_Num을 이용하여 마지막 T1의 위치(LoT1 : Last of TrailingOnes) 이후 '1'로 채워진 Level Mask를 생성한다. 그리고 Level Mask를 Non-zero flag와 AND 연산하여 Level Flag를 구한다. Bitstream Checker는 '0'이 아닌 계수 사이에 존재하는 연속된 '0'의 개수(Run Before)를 찾기

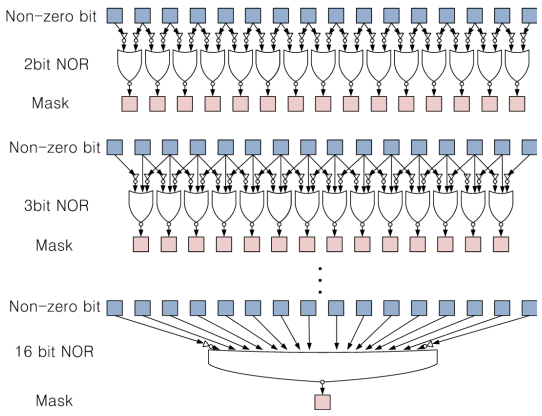


그림 13. 비트열 검출기
Fig. 13. Bitstream Detector.

위해 그림 13의 비트열 검출기를 이용하여 Non-zero flag를 2~14비트까지 분할하여 “11, 101, 1001, …, 1000000000000001”의 비트열에 대하여 NOR 논리 연산하여 그림 10의 flag 행렬을 만든다. Run Before Encoder는 Bitstream Checker의 수직 15비트열을 표 3의 부호화 테이블을 이용하여 계수들의 Run Before를 출력한다. 그리고 Bitstream Checker의 수직 15비트열을 OR연산하여 Run Before 여부를 나타내는 Run Flag를 구한다.

2. 코드워드 부호화 블록

그림 14의 구조를 갖는 코드워드 부호화 블록은 TC

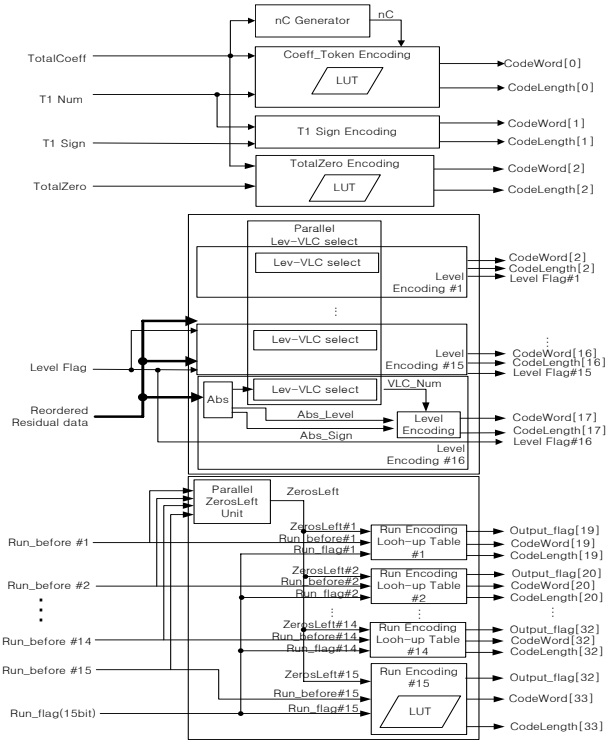


그림 14. 코드워드 부호화 블록의 구조
Fig. 14. Codeword coding block architecture.

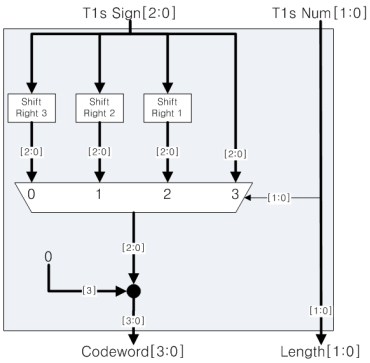


그림 15. T1_sign 부호화 모듈
Fig. 15. T1_sign encoding module.

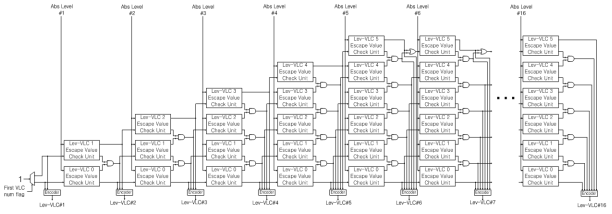
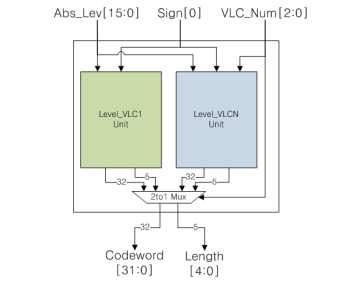
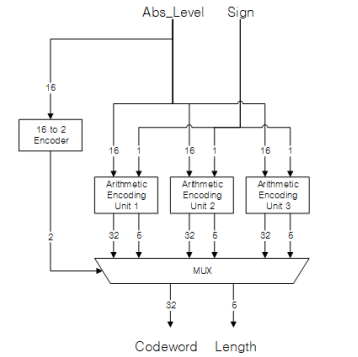


그림 16. 병렬 레벨 값 연산기
Fig. 16. Parallel Lev-VLC selector.

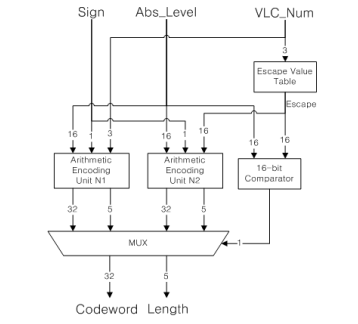


(a) Level encoding module



- Arithmetic Encoding Unit 1 (Abs_Lev < 8)
  - Codeword = 1
  - Length = (Abs_Lev <<1) + sign - 1
- Arithmetic Encoding Unit 2 (8 ≤ Abs_Lev < 16)
  - Codeword = 16 | ((Abs_Lev - 8) << 1) | sign
  - Length = 19
- Arithmetic Encoding Unit 3 (16 ≤ Abs_Lev)
  - Codeword = 4096 | ((Abs_Lev - 8) << 1) | sign
  - Length = 28

(b) Level VLC1 unit



- Arithmetic Encoding Unit N1 (Abs_Lev < Escape)
  - Codeword = (1<<VLC_Num) | ((Abs_Lev - 1) & ~(0xFFFFFFFF<<(VLC_Num - 1)))<<1 | sign
  - Length = ((Abs_Lev – 1)>>(VLC_Num – 1)) + VLC_Num + 1
- Arithmetic Encoding Unit N2 (Abs_Lev ≥ Escape)
  - Codeword = 4096 | (Abs_Lev – escape)<<1 | sign
  - Length = 28

(c) Level_VLCN unit

그림 17. 레벨 산술 부호화 모듈
Fig. 17. Level Arithmetic encoding module.

를 입력받아 nC연산을 수행하고 Look-up table을 선택하여 T1_Num를 이용해 CT 부호화를 수행한다. 그와 동시에 T1_sign 부호화 모듈에서는 그림 14와 같이 T1_sign과 T1_num을 입력받아 T1_sign을 각각 shift 한 후 T1_num에 따라 선택하여 출력한다. 이때 T1_num는 코드길이로서 같이 출력된다. TZ 부호화 모듈에서는 TC와 TZ를 입력받아 Look-up table을 이용하여 부호화를 수행한다.

레벨의 코드워드 부호화에서는 잔여화소와 Level flag를 입력 받아 그림 16의 병렬 레벨값 연산기를 통해 16개의 레벨 값(Lev VLC_num)을 연산하고, 이후 그림 16 (a)의 레벨 산술 부호화 모듈에서 레벨 값과 레벨에 따라 산술 부호화를 수행한다. 병렬 산술 부호화기는 변환계수의 절대값과 임계값을 비교한다. 계수의 절대값이 임계값보다 클 경우 flag신호를 출력하고 flag신호를 인코딩하여 16개 레벨 값(Lev-VLC num)을 동시에 구한다. 레벨 부호화는 그림 17에서와 같이 레벨 값이 '0'이면 Level_VLC1 unit를 통해 산술부호화하고, 레벨 값이 '0'이 아니면 Level_VLCN unit를 통해 산술 부호화한다. 각 부호화 모듈과 산술연산수식은 그림 17의 (b)와 (c)에 나타내었다. 레벨의 코드워드 부호화가 완료된 후에는 Level flag에 따라 선택적으로 부호화 결과를 사용할 수 있도록 flag신호와 함께 출력한다.

Run 부호화는 스캔과정에서 구한 Run Before값을 모두 합하여 ZeroLeft값을 구한다. 그림 18와 같이 병렬 입력되는 Run Before를 하나씩 오른쪽으로 이동하며 덧셈하면 각 위치에서의 ZeroLeft를 동시에 구한다. 그리고 ZeroLeft값을 참조하여 룩업테이블을 선택하면, Run Before값을 부호화 할 수 있다.

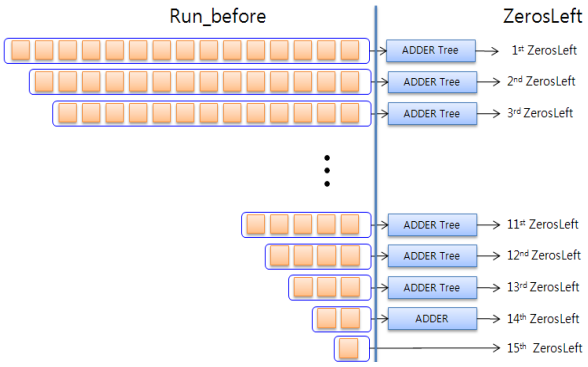


그림 18. Parallel ZerosLeft 연산
Fig. 18. Parallel ZerosLeft Processing.

Ⅳ. 실험 및 고찰

제안하는 CAVLC 부호화기를 Verilog HDL로 구현하여 0.13㎛ 표준 셀 라이브러리 환경에서 시뮬레이션 하였다. 표 3은 기존 구조와 제안하는 CAVLC 부호화기의 성능 비교한 결과이다. CAVLC 부호화기의 연산처리성능(Throughput)을 비교하기 위해 매크로 블록당 처리 사이클을 비교하였다. 통계값 탐색 과정과 코드워드 부호화 과정을 각 1사이클씩 처리하는 제안하는 구조는 매크로 블록을 처리하는데 29사이클 걸린다. 연산처리성능이 가장 뛰어난 연구[6]가 매크로 블록을 처리하는데 110사이클로 걸리며, 제안하는 구조와 비교하면 3.5배 이상의 처리 성능이 뛰어남을 알 수 있다. 면적을 비교하기 위해 게이트 수를 비교하면, 제안하는 구조의 게이트 수는 33.4Kgates이다. 처리성능이 가장 좋은 기존 연구의 게이트수가 15Kgates인 것에 비하면 면적이 약 2.3배 증가하였음을 알 수 있다. 따라서 면적 대비 처리 성능이 훨씬 개선되는 결과를 보임을 알 수 있다. 최대 동작주파수는 100MHz이며, UD(3840×2160) 동영상을 초당 100프레임으로 처리하는 결과이다. 이는 Full HD 동영상을 초당 60프레임으로 처리하는 기존 연구보다 4배 이상 향상된 결과이다.

표 3. CAVLC 성능 비교표
Table 3. CAVLC Performance Comparison.

구분	[2]	[3]	[4]	[5]	[6]	[7]	[8]	제안 구조
구현 환경 (㎛)	0.35	0.18	0.18	0.18	0.18	0.18	0.18	0.13
연산처리 성능 (Cycle/MB)	-	432	500	450	110	256	118	29
Area (Kgates)	9.1	9.7	23.6	16.4	12.1	15	17	33.5
최대동작 주파수 (MHz)	66	125	100	100	125	125	125	100
처리성능 (해상도@fps)	176x 144 @10	1920x 1080 @30	1920x 1080 @30	1920x 1080 @30	1920x 1080 @30	1920x 1080 @60	-	3840x 2160 @100

Ⅴ. 결 론

본 연구에서는 UD의 초고해상도 동영상을 실시간 처리하는 고성능 H.264/AVC CAVLC 부호화기를 설계하고자하였다. 고성능 H.264/AVC CAVLC 부호화기를

구현하기 위해 통계값 탐색 과정과 코드워드 부호화 과정을 1사이클에 처리하도록 처리성능을 향상시켰다. 먼저 16개 계수가 '0'인지 표기하는 플래그 형태의 16비트 열을 만들어, 산술 및 논리연산을 통해 1사이클에 통계값을 구하였다. 그리고 레벨의 코드워드 부호화 과정에서 발생하는 데이터 종속성을 제거하여 순환 연산이 없이 1사이클에 코드워드 부호화 과정을 처리하였다. 제안하는 CAVLC는 0.13㎛ 표준 셀 라이브러리 환경에서 시뮬레이션 하였다. 제안하는 CAVLC 부호화기의 면적은 기존 구조 대비 2.3배 증가하였으나, 데이터 처리성능은 기존 구조에 비해 4배 향상된 결과를 보여 면적 증가 대비 처리성능이 개선됨을 보인다. 제안하는 CAVLC 부호화기의 최대 동작주파수는 100MHz이며, UD 동영상을 초당 100프레임으로 처리하여 기존 연구보다 4배 이상 향상되었음을 확인하였다.

참 고 문 헌

[1] Evaluation and Simplification of H.26L Baseline Coding Tools. M. Zhou. JVT-B030, Jan,2002.

[2] Yeong-Kang Lai; Chih-Chung Chou; Yu-Chieh Chung; "A Simple and Cost Effective Video Encoder with Memory-Reducing CAVLC", ISCAS 2005, IEEE International Symposium on Volume 1, 18-20, Sept. 2003, pp. 323-326

[3] A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications. Chih-Da Chien, Keng-Po Lu, Yi-Hung Shih, Jiun-In Guo.

[4] Architecture design of context-based adaptive variable length coding for H.264/AVC. Tung-Chien Chen; Yu-Wen Huang; Chuan-Yung Tsai; Bing-Yu Hsieh; Liang-Gee Chen. Circuits and Systems II: Express Briefs, IEEE Transactions on Volume: 53 , Issue: 9 TCSII.2006.880014 Publication Year: 2006, Page(s): 832-836

[5] 실시간 HD급 영상 처리를 위한 H.264/AVC CAVLC 부호화기의 하드웨어 구조 설계. 우정욱, 이원재, 김재석. 2007년 7월 전자공학회 논문지 제 44 권 SD 편 제 7 호.

[6] Highly efficient CAVLC encoder for MPEG-4 AVC/H.264. T.-H. Tsai S.-P. Chang T.-L. Fang. Circuits, Devices & Systems, IET Volume 3,Issue 3, June 2009 Page(s):116-124.

[7] Forward Computations for Context-Adaptive Variable-Length Coding Design, Shih-Chang Hisa, Wen-Hsien Liao, August, 2010.

[8] H.264/AVC를 위한 고성능 CAVLC 부호화기 하드웨어 설계, 이양복, 류광기. 2012년 3월 전자공학회 논문지 제 49 권 SD 편 제 3 호.

저 자 소 개

고 병 수(정회원)

2003년 광운대학교 컴퓨터공학과 학사 졸업.

2005년 광운대학교 컴퓨터공학과 석사 졸업.

2005년 ~ 현재 광운대학교 컴퓨터공학과 박사과정

<주관심분야: 영상신호처리 SoC설계, Embedded Computing System 설계>

공 진 흥(평생회원)-교신저자

1980년 서울대학교 전자공학과 학사 졸업.

1982년 한국과학기술원 전기 및 전자공학과 석사 졸업.

1989년 텍사스주립대학교 컴퓨터공학과 박사 졸업.

1989년 ~ 현재 광운대학교 컴퓨터공학과 교수

<주관심분야: 영상신호처리 SoC설계, Embedded Computing System 설계>