논문 2008-45SD-11-6

새로운 H.264/AVC CAVLC 고속 병렬 복호화 회로

(A New H.264/AVC CAVLC Parallel Decoding Circuit)

여 동 훈*, 신 현 철**

(Donghoon Yeo and Hyunchul Shin)

요 약

새로운 컨텍스트 기반 적응형 가변 길이 코드의 효율적인 병렬처리 기법을 개발하였다. 본 논문에서는 확장적인 병렬처리, 작은 면적, 저전력 설계를 위한 몇 가지 새로운 아이디어 제시한다. 첫 번째, 빠른 저전력 연산을 위해 메모리 방식 대신에 단순화된 논리 연산 방식으로 회로를 설계하였다. 두 번째, 효율적인 논리 연산을 위하여 코드 길이를 이용하여 코드들을 그룹지었다. 세 번째, M 비트까지의 입력은 고속 처리를 위하여 병렬 처리하였다. 비교를 위해 M=8인 병렬 논리 연산 복호기와 대표적인 기존 방식의 복호기를 설계하여 비교하였다. 실험 결과, 제안한 기법은 고속 병렬처리가 가능하며, 같은 복호 속도 (M=8일 때, 1.57codes/cycle) 에서는 기존 방식의 복호기보다 46% 작은 면적을 사용한다.

Abstract

A new effective parallel decoding method has been developed for context-based adaptive variable length codes. In this paper, several new design ideas have been devised for scalable parallel processing, less area, and less power. First, simplified logical operations instead of memory look-ups are used for fast low power operations. Second the codes are grouped based on their lengths for efficient logical operation. Third, up to M bits of input are simultaneously analyzed. For comparison, we have designed the logical operation based parallel decoder for M=8 and a typical conventional method based decoder. High speed parallel decoding is possible with our method. For similar decoding rates (1.57codes/cycle for M=8), our new approach uses 46% less area than the typical conventional method.

Keywords : Parallel decoding, CAVLC, CAVLD, H.264

I. 서 론

다양한 멀티미디어 서비스와 모바일 기기의 대중화로 인하여 표준화가 필요하였고, ITU-T/ISO/IEC Joint Video Team에 의해 동영상 표준인 H.264/AVC가 제정되었다[1]. H.264/AVC에서 사용하는 가변 길이 코드는 발생 확률이 높은 입력 심볼에 길이가 짧은 코드를 인가하고, 발생 확률이 낮은 경우에 길이가 긴 코드를 인가하는 코드 압축 기법이다. H.264/MPEG-4 AVC에서는 2가지의 개선된 가변 길이 부호화 기술, 즉 컨텍스트 기반 적응형 가변 길이 부호화 (Context-based adaptive variable length coding:CAVLC) 와 컨텍스트 기반 적응형 산술 부호화 (Context-based adaptive binary arithmetic coding:CABAC) 를 채택하고 있다. 컨텍스트 기반 적응형 가변 길이 부호화는 5가지의 신텍스(Syntax) 타입을 가지고 있으며, 5가지 신텍스 타입은 Coeff_token, Trailing_ones, Level, Total_zeros, 그리고 Run_before이다. 일반적으로 컨텍스트 기반 적응형 가변 길이 복호기는 룩업 테이블을 사용함으로 인해 코드워드가 검색되기까지 다중 메모리 액세스가 필요하다. DMB 플레이어, PMP, PDA, 그리고 모바일폰과 같은 동영상 처리 기능이 내장된 멀티미디어 기기들은 연산과정에서의 많은 전력 소모와 지연시간을 발생

* 정회원, 한양대학교 메카트로닉스공학과
(Mechatronics Engineering, Hanyang University)
** 평생회원, 한양대학교 전자전기제어계측학과
(Electronic Engineering and Computer Science, Hanyang University)
※ 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구 결과로 수행되었음 (IITA-2008-(C1090-0804-0009)).
접수일자: 2008년8월28일, 수정완료일: 2008년10월30일

시키는 상당한 메모리 액세스를 요구한다[2~3].

본 논문에서는 새로운 복호화 논리 연산 기법을 소개한다. 논리 연산 기법은 5가지의 모든 신텍스 타입에 적용된다. 논리 연산 기법은 산술 연산 기법에 비해 동작이 단순하며 따라서 전력 소모가 감소한다. Ⅱ장에서는 메모리 참조, 산술 연산, 논리 연산을 적용한 전형적인 설계들을 비교 설명한다. Ⅲ장에서는 병렬 컨텍스트 기반 적응형 가변 길이 복호화에 관하여 설명한다. Ⅳ장에서는 실험을 통한 결과를 보이며, 마지막으로 Ⅴ장에서는 결론으로 마무리한다.

Ⅱ. 전형적인 복호기 설계 기법

1. 컨텍스트 기반 적응형 가변길이 부호화

H.264/AVC의 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 무손실 압축하기 위한 알고리즘이다. 이 블록의 화소 값들은 CAVLC 알고리즘으로 복호화 된다. 다음은 CAVLC의 복호화 5단계를 설명한다.

(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. 해당 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 지그-재그 스캔의 역순으로 복호화가 진행된다. 레벨의 값들을 복호하기 위해서는 7가지의 룩업 테이블을 사용한다.

(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 있는 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 1.에서 CAVLC의 예를 설명한다. 다음 절에서 이 예를 이용하여 3가지 기존 기법의 특징을 설명한다. 그림에서 4x4 DCT 상수와 Coeff_token, Trailing_ones, Level, Total_zeros, 그리고 Run_before의 코드들을 볼 수 있다[5]. 지그-재그 스캔을 통해서 0, -3, 0, 1, -1 ,-1, 0, 1이 순서로 추출된다. Total_coeff는 5(-3, 1, -1, -1, 1)이고, Total_zeros는 3이며, Trailing_ones는 3(-1, -1, 1)이다. 사실 4개의 Trailing_ones가 있지만 오직 3개만 부호화하고 나머지 1개는 Level 코드로 부호화한다. Run_before는 zero_left가 0이 될 때까지 반복된다. 비트스트림은 그림 2에 보인 바와 같이 전송된다.

0	-3	-1	0
0	-1	1	0
1	0	0	0
0	0	0	0

Coeff_token	0000100
Trailing_ones(4)	0
Trailing_ones(3)	1
Trailing_ones(2)	1
Level(1)	1
Level(0)	0010
Total_zeros	111
Run_before(4)	10
Run_before(3)	1
Run_before(2)	1
Run_before(1)	01
Run_before(0)	No code required

The bitstream for this block is
000010001110010111101101

그림 1. 컨텍스트 기반 적응형 가변 길이 부호화의 예
Fig. 1. CAVLC for a 4x4 block.

2. 메모리 참조 기법

일반적인 VLC 복호기는 Look-up Table(LUT)이 메모리로 이루어져 있어 많은 횟수의 메모리 액세스가 필요하다[2~3]. 메모리 참조 기법에서 코드워드는 참조 테이블의 주소로 쓰여진다. Lool-up table은 다수의 작은 부분으로 파티션 된다. 발생 빈도가 높은 코드는 적은 색인의 Look-up table에 위치한다. 현재의 Look-up

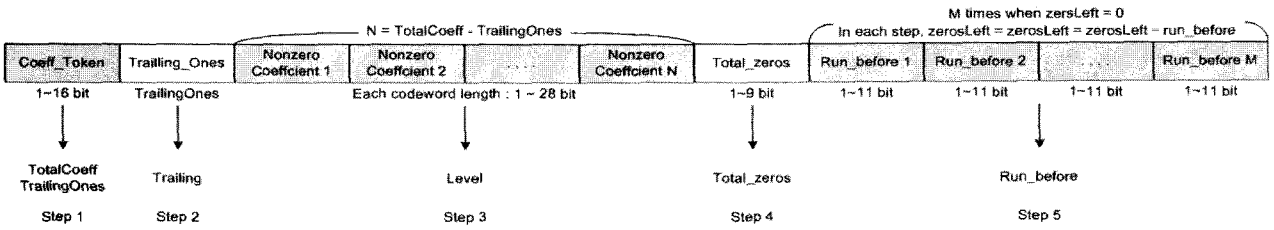


그림 2. CAVLC 복호화 비트스트림의 구조
Fig. 2. Bitstream of CAVLC decoding.

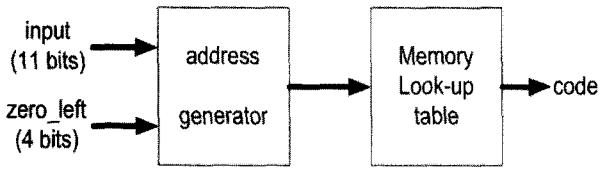


그림 3. Look-up table에 의한 Run_before 복호화
Fig. 3. Run_before decoding by look-up table.

table에 매칭되는 코드워드가 검색되지 않을 때에는 다음 Look-up table와 매칭을 한다. 그림 3에서는 언급한 Look-up tables로 그림 1 예의 Run_before값을 찾는 것을 나타낸다. 이 기법의 경우 메모리 크기를 줄이기 위해 메모리 분할 기법이나 산술 연산 기법을 사용하지만 코드 길이 확인 후에 산술 연산을 수행하므로 병렬처리가 어렵다.

3. 산술 연산 기법

메모리 액세스로 인해 발생하는 전력 소모를 줄이기 위한 방법으로 [3]에서는 CAVLC 복호기의 Run_before 단계에서 산술 연산으로 구성된 FSM 기법을 제안하여 Run_before 단계에서의 메모리 액세스를 제거하였다. 다음은 [3] 기법을 그림 1의 예에 적용한 것이다.

Run_before(4) : zero_left - Run_before(4)[1:0]
= 3 - '10' = 1

Run_before(3) :
(2-Run_before(3)[1:0]) • (1-Run_before(3)[1])
= (2 - '1') • (1 - '1') = 0

Run_before(2) :
(2-Run_before(2)[1:0]) • (1-Run_before(2)[1])
= (2 - '1') • (1 - '1') = 0

Run_before(1) :
1 - Run_before(1)[1] = 1 - '0' = 1

이 방법의 단점은 Run_before 단계를 제외한 나머지 단계들에서의 메모리 엑세스는 여전히 존재하고, 직렬 복호 방식이라서 병렬 복호 방식보다 처리 속도가 느리다는 것이다. 또한 산술 연산을 위해 코드 길이 확인이 필요하여 병렬처리가 어렵다.

4. 복호화 기반 논리 연산 기법

본 연구에서는 효과적인 하드웨어 동작을 위해 앞에서 언급한 메모리 look-up 방식, 산술연산 방식 대신 논리연산 방식을 사용하였다. 논리연산 방식은 간단하고 빠르며 보다 적은 전력을 소모한다. 그림 4는 그림 1의 회로에서 논리연산을 이용한 Run_before 코드의 복호화 부분을 나타낸 것이다. 대부분의 회로가 메모리 look-up 방식에 비하여 매우 간단하다.

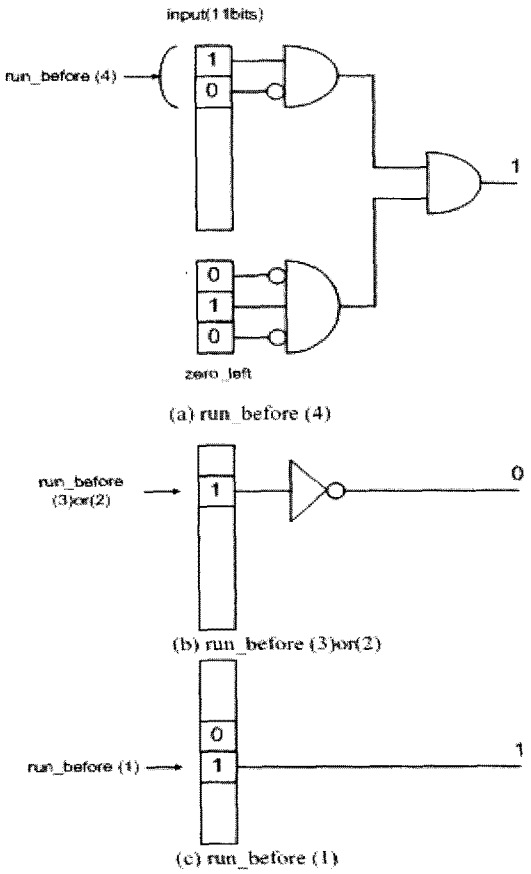


그림 4. 논리연산자를 이용한 Run_before
Fig. 4. Run_before decoding by logical operation.

5. Multiple-Symbol Parallel Variable Length Decoding 기법

가변길이코드는 코드 길이의 가변성으로 인하여 병렬처리를 효과적으로 수행하기 어렵다. [6]에서는 MPEG-2 기반의 VLC를 효과적으로 병렬처리 할 수 있는 multiple- symbol parallel variable length decoding 기법을 제시하였다. 그림 5는 [6]에서 제안한 원리를 설명한 그림이다. 각 비트에서 시작되는 발생가능한 모든 대응 코드를 검색한 후, 첫 매칭 된 코드의 다음 비트에서 시작하는 코드를 선택한다. 이 과정을 반복해서 N-Bit의 모든 입력을 복호할 수 있는 병렬처리가 가능하다.

그림 6은 multiple-symbol parallel variable length decoding 구조이다. 코드의 길이가 2이상이므로, 2번째 최상위 비트부터의 코드 탐색은 불필요하다. 따라서 Codeword Detector (CD)의 개수는 입력 스트림 비트 수를 N이라 하면 (N - 2) 가 된다. 그림 6에서는 16비트의 입력 스트림을 가정하므로, 14개의 CD가 필요하게 된다. 따라서 N - 2 의 CD로 인하여 면적 증가가 매우 심하다. 뿐만 아니라 CAVLC는 5단계의 코드로 나누어지므로, 이 구조는 CAVLD에 적합하지 못하다.

III. 새로운 병렬 디코더

1. 제안한 기법의 알고리즘

앞에서 설명한 것과 같이 1개의 code length detector를 사용할 경우 병렬처리가 어렵고, 여러 개의 CD를 사용하여 병렬처리 하면 처리 속도는 빠르지만 면적이 크게 증가하는 문제점과 LUT 엑세스로 인한 전력 소모의 문제점을 갖고 있다. 따라서 본 연구에서는 병렬처리에 효과적이고 메모리 액세스를 제거한 새로운 Extensive Parallel Decoding 기법을 개발하였다. Extensive Parallel Decoding의 핵심은 코드의 그룹화와 M개의 입력 비트까지의 병렬처리이다. 여기서 M은 설계자가 원하는 병렬처리 정도에 따라 정하는 자연수이다. VLC에서는 정해진 규칙에 따라 코드가 생성된다. 따라서 일정 길이 안에서 존재하는 모든 코드들을 하나로 묶어 그룹을 만들 수 있다. 입력 스트림의 코드가

S	C_k	m_k
A	10	2
B	11	2
C	011	3
D	0100	4
E	0101	4
F	00101	5
G	00110	5
H	000110	6
I	000111	6
J	0000110	7
K	00100101	8

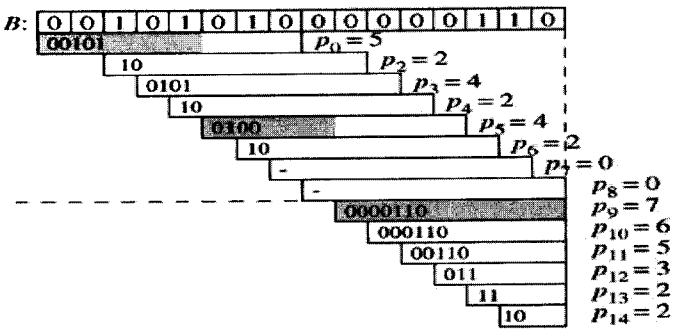


그림 5. Multiple-symbol parallel processing기법의 예제 코드 테이블[6]
Fig. 5. Example code table of multiple-symbol parallel processing[6].

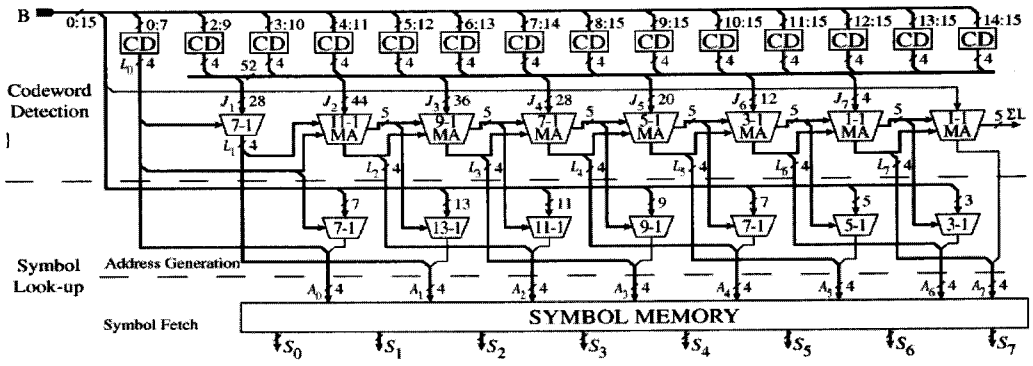


그림 6. Multiple-symbol parallel variable length decoding 구조[6]
Fig. 6. Multiple-symbol parallel variable length decoding.

$C_1, C_2, C_3, \cdots, C_i, \cdots$라 하고, 코드 C_i의 길이를 l_i라 하면, M개의 입력 비트까지의 병렬처리는 다음과 같이 이루어진다.

i) $l_1 \leq M$ 일 때

$L = \sum_{i=1}^{k} l_i \leq M$인 최대 정수 k를 구한다. k개의 연속된 코드들은 $(C_1, \cdots, C_k)$으로 하나의 그룹 G_L을 형성한다. 위와 같이 구성된 G_L은 병렬처리 복호화 하여 한 클락에 k개의 코드를 복호화 한다.

ii) $l_1 > M$ 일 때, $L = l_1$이고 $k = 1$이다. 즉, 코드 C_1 하나만을 한 클락에 복호화 한다.

그림 7에서는 M이 8일 때, 제안한 알고리즘의 동작을 예로 설명한 것이다. 첫 번째 코드와 두 번째 코드까지의 길이 합이 5이므로 코드 그룹 G_5이 되어 2개의 코드들을 한 번에 복호화 한다. 다음 사이클에서 세 번째 코드와 네 번째 코드 길이 합이 8이므로 코드 그룹은 G_8이 되고, 병렬처리 된다. 다음 사이클에는 다섯 번째 코드 길이가 10비트이므로 하나의 코드(G_{10})만 복호화 된다.

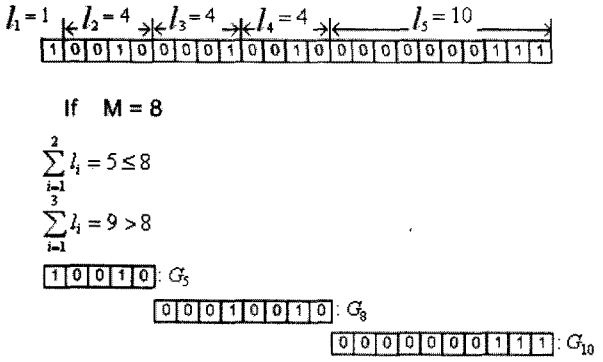


그림 7. Extensive Parallel Decoding의 예
Fig. 7. Example of Extensive Parallel Decoding.

2. CAVLD에 적용한 Extensive Parallel Decoding

CAVLC 복호화는 앞장에서 설명한 5단계(Coeff_token, Trailing_ones, Level, Total_zeros, Run_before)로 순차적으로 처리된다. 현 단계가 완료되어야 다음 단계를 처리할 수 있다. 따라서 각 단계가 서로 독립적이지 못하기 때문에, 다른 단계들 간에는 동시적으로 병렬처리를 할 수 없다. Coeff_token, Total_zeros 단계는 하나의 매크로 블록 복호화 과정에서 한 번씩 발생하지만, Trailing_ones, Level, Run_before 단계는 여러 번 발생할 수 있다. 동일 단계의 코드들도 순차적으로 처리되므로, 서로 간에 독립적이지 못하다. 하지만 동일 단계에서는 수행 과정이 같다는 점과 선행 코드가 다음 코드에 영향을 미친다는 점을 이용하여, 제안한 기법을 M이 8일 때 CAVLC 복호기에 적용하였다.

제안한 기법은 일정 길이 내의 존재할 수 있는 코드들을 하나의 그룹으로 묶는 것이므로 코드의 종류가 많을수록 코드들의 다양한 집합이 발생한다. 가변적인 코드 길이로 인한 코드 검색 오류를 막기 위해 VLC는 일정한 패턴을 가지고 있다. 따라서 발생할 수 없는 코드들의 집합을 제외하면 코드 그룹의 수는 많이 줄어든다.

가. Coeff_token

Coeff_token단계는 Total_coeff와 Trailing _ones를 출력한다. Coeff_token단계는 매크로블록의 복호화 과정 중 최초 단계로서 단 1회만 복호화 된다.

나. Trailing_ones

Trailing_ones단계는 전 단계인 Coeff_token단계에서

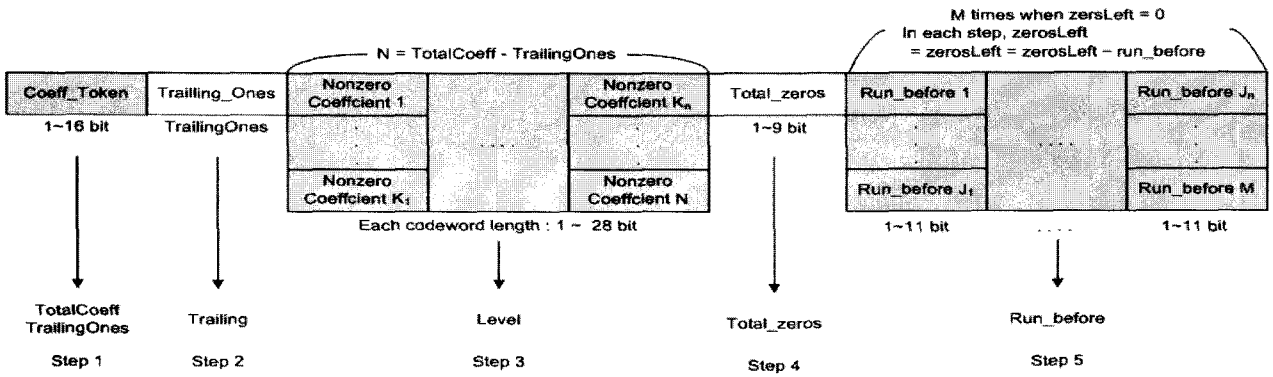


그림 8. Parallel processing CAVLD 방법에 사용할 비트스트림 구조
Fig. 8. Bitstream of Parallel processing CAVLD.

출력된 Trailing_ones값을 이용하여 복호화 과정을 수행한다.

다. Level

Level단계는 Level_VLC0에서 Level_VLC6까지의 7개의 LUT가 있다. 첫 번째 Level은 Level_VLC0을 엑세스한다(Total_coeff가 10보다 크고 Trailing_ones가 3보다 작으면 예외적으로 Level_VLC1을 엑세스한다). 복호한 Level 계수의 크기가 정해진 임계값보다 크면 다음 Level_VLC 테이블로 넘어간다. 임계값보다 같거나 작으면 현재의 Level_VLC 테이블을 유지한다. 이것은 하나의 코드를 알면 그 다음의 코드가 어느 테이블에 해당하는지 예상할 수 있다는 것을 의미한다.

라. Total_zeros

Total_zeros단계는 최상위 0이 아닌 상수까지의 0의 개수를 복호화 한다. Coeff_token단계의 Total_coeff값, 코드워드 및 Total_zeros 단계 선택 신호를 Logic operation하여 Total_zeros를 복호화 한다.

마. Run_before

Run_before 단계는 Run_before의 총합이 Total_zeros와 같을 때까지 진행된다. zero_left는 복원되지 않고 남은 0의 개수를 나타낸다. 초기 시작 zero_left는 선행단계인 Total_zeros의 값에 따라 결정된다. Run_before에서도 Level과 같이 현재 복호화 된 코드에 의해 다음 복호화 되는 코드의 조건이 결정된다. 이 점을 이용하면 Level과 같은 방법으로 M Bit 내 존재하는 코드 그룹을 결정할 수 있다. 그림 1의 예에서 Total_zeros는 2이고 Run_before도 2이므로 Total_zeros와 Run_before가 같아서 Run_before 단계는 1회만 실행된다. [2]에서 제안한 기법을 적용하면 위 예제를 실행 시 4사이클이 소모되지만, 본 논문에서 제안한 Extensive Parallel Decoding을 적용하면 1사이클에 Run_before를 수행할 수 있다.

Level 단계에 비하여 Run_before 단계 코드는 코드 길이가 짧은 것들이 많으므로 Level 수행에 비해 Run_before 수행 시 제안한 기법으로 얻을 수 있는 이득이 더 크다. 그림 1의 예에서 제안한 기법으로 Run_before 단계는 1 사이클에 4개의 코드들을 병렬처리 하지만 Level 단계는 2개의 코드들을 복호한다. 따라서 Run_before 수행 횟수가 많은 경우에 Extensive Parallel Decoding은 더 좋은 성능을 보인다. 그림 9는

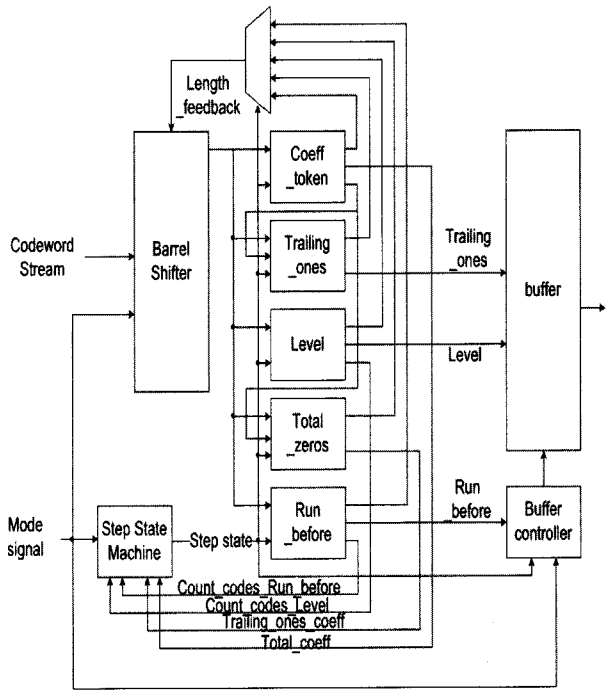


그림 9. 제안한 기법을 적용한 CAVLC 복호기 블록도
Fig. 9. Logical operation based parallel CAVLC decoder.

제안한 기법을 적용한 CAVLC 복호기 블록도를 나타낸 것이다.

3. Prefix precomputation을 이용한 제안한 기법의 개선

앞 절에서의 제안한 기법을 적용한 설계는 각 모듈의 모든 입력 비트들을 논리 연산 처리하기 때문에 불필요한 면적 증가가 있다. 따라서 기존의 2장에서 설명한 메모리 참조 기법을 응용한 설계 기법을 적용하면 면적을 더욱 감소시킬 수 있다.

메모리 참조 기법에서는 주된 연구 주제가 메모리의 크기를 최소화 시키는데 있다. 따라서 메모리 파티션에 관한 연구가 메모리 참조 기법의 주류를 이루고 있다. 메모리 파티션을 통해 코드들의 prefix 부분을 별도로 디코딩하여 메모리의 크기를 줄인다.

제안한 병렬디코더의 면적을 줄이기 위하여 prefix

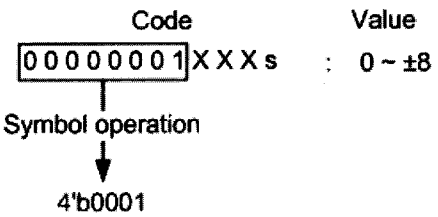


그림 10. Symbol Logic operation의 예
Fig. 10. Example of Symbol Logic operation.

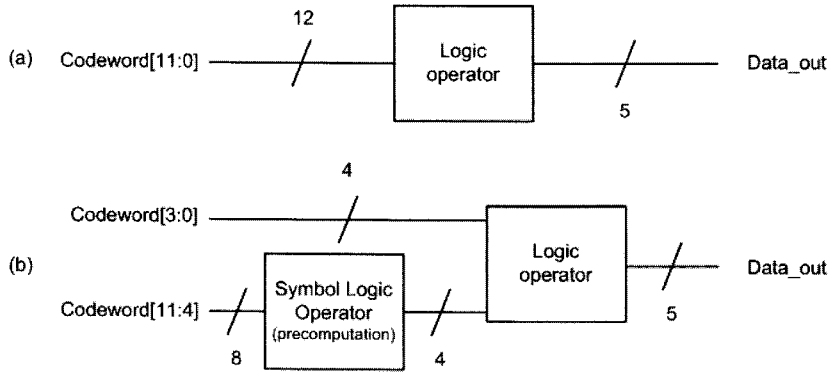


그림 11. Symbol Logic operation을 적용한 설계 비교
Fig. 11. Comparison of design with Symbol Logic operation

표 1. Prefix의 Symbol Logic operation
Table 1. Symbol Logic operation of prefix.

Prefix	Symbol Logic operation	
00001	A	4'b0000
000001	B	4'b0001
0000001	C	4'b0010
00000001	D	4'b0011
000000001	E	4'b0100
0000000001	F	4'b0101
00000000001	G	4'b0110
000000000001	H	4'b0111
0000000000001	I	4'b1000
00000000000001	J	4'b1001
000000000000001	K	4'b1010
0000000000000001	L	4'b1011

부분을 precomputation하여, Logic operation 회로의 입력 수를 감소시켰다. 그림 10은 코드 길이가 12이고 Prefix의 길이가 8비트인 코드들로 입력이 이루어진다고 가정하여, 간략하게 Symbol Logic operation을 설명한 것이다. 코드의 Prefix 부분인 '00000001'을 '0001'로 Symbol화 하면 실질적인 Logic operation의 입력 신호는 12비트에서 8비트로 감소하게 된다.

그림 11은 Symbol Logic operation을 적용한 후의 구조를 설명한 것이다. 그림 11.(a)는 Symbol Logic .operation을 적용하기 전의 구조이고 그림 11.(b)는 적용 후의 구조이다. 적용 후 구조에서는 Prefix 부분을 Symbol Logic operation 해주는 8-input 게이트가 존재하며 실질적인 Logic operator의 입력은 Prefix 부분을 Symbol화한 4비트 신호와 Subfix 부분의 4비트로 구성된 8비트 신호이다. 그림 11의 예에서 본 바와 같이 실질적인 입력 비트 수의 감소로 인하여 Symbol Logic operation 적용 전 구조는 33개의 인스턴스들 (gates)로 구성되지만 적용 후 구조는 23개의 인스턴스들 (gates) 로 구성된다. 또한 인스턴스의 종류 또한 적용 전 구조에서는 input 비트가 큰 게이트들을 많이 사용한다.

입력 비트들의 많은 부분을 차지하는 입력코드의 비트를 표 1과 같이 Symbol화를 하여 입력을 처리하는데 필요한 게이트의 수를 줄임으로써 면적을 추가적으로 약 11% 감소시킬 수 있었다.

IV. 실 험 결 과

1. Multiple-Symbol Parallel Decoding의 CAVLC 적용

CAVLC 복호기에 특화된 병렬처리의 경우는 CAVLC의 알고리즘 특성을 이용한 것이므로 다른 VLC 복호기에 적용하는 것이 용이하지 않다. 따라서 본 연구에서는 Multiple-Symbol Parallel Decoding[6] 기법과 제안한 VLC 복호화 병렬처리 기법의 성능과 면적을 비교하였다. 성능 비교를 위해 JM 10.2 인코더를 사용하여 QP 24로 Foreman영상을 부호화 하였다. 표 2는 Multiple-Symbol Parallel Decoding[6] 기법을 적용한

표 2. 제안한 기법의 성능 비교
Table 2. Performance comparison.

	Multiple-Symbol Parallel Decoding[6]		제안한 기법 (Extensive parallel)
Throughput (Foreman : QP 23)	CD #6	1.56code/cycle	1.57 code/cycle
	CD #7	1.59code/cycle	
	CD #8	1.63code/cycle	
	CD #32	1.98code/cycle	

CAVLC 복호기와 우리의 Parallel Decoding CAVLC 복호기의 성능을 비교한 표이다. [6]을 적용한 CAVLC 복호기는 Codeword Detecter(CD)를 6, 7, 8, 32개를 사용했을 경우와 제안한 기법이 8 Bit까지 병렬처리 할 경우의 성능을 비교하였다.

2. 제안한 기법과 Multiple-Symbol Parallel Decoding을 적용한 CAVLC의 성능 비교

표 3.은 Verilog HDL으로 직접 설계하고 Hynix 0.25um 공정 라이브러리를 적용하여 Synopsys Design Analyzer로 면적을 비교한 것이다. 표 3에서 제안한 기법의 면적 크기는 N이 6일 때 [1] 기법의 면적 크기에 비해 40% 감소하였다. 면적에서의 이득은 N이 7, 8일 때의 성능 차이를 감안하여도 상당한 이점이 있다.

표 3은 Hynix 0.25um 공정 라이브러리를 적용하여 Synopsys Design Analyzer로 합성한 결과를 나타낸다. 설계 기법 개선을 통해 개선 전에 비하여 약 11%의 면적이 감소하였고, Multiple-Symbol Parallel Decoding[6] 기법에 비하여 같은 성능 (CD #6) 에서 면적은 46% 감소하였다.

표 3. 제안한 기법의 면적 비교
Table 3. Area comparison.

	Multiple-Symbol Parallel Decoding[6]		개선 전 제안 기법	개선 후 제안 기법
Area (um x um)	CD #6	233928	139252	124847
	CD #7	263019		
	CD #8	287384		
	CD #32	615867		

V. 결 론

H.264에서 사용하는 Variable Length Coder/Decoder (VLC/VLD) block을 설계하였으며, 효율적인 설계를 통해 기존 방법에 비해 면적과 크기 면에서 우수한 반도체 회로를 구현 할 수 있었다. 제안한 VLC/VLD block은 기존의 memory table을 사용하던 방식과는 다르게 논리회로 방식과 Prefix precomputation 방법을 사용하여 같은 복호화 속도에서 면적을 기존 병렬처리에 비하여 46% 감소시켰다.

참 고 문 헌

[1] "Joint Final Committee Draft (JFCD) of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)," Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, 4th Meeting, Klagenfurt, Austria, 2002.

[2] Y. Kim, Y. Yoo, J. Shin, B. Choi, and J. Paik, "Memory-Efficient H.264/AVC CAVLC for Fast Decoding," IEEE Transactions on Consumer Electronics, Vol. 52, No. 3, 2006.

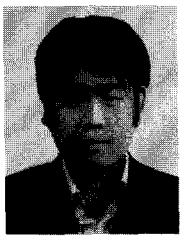
[3] Y. Moon, G. Kim, and J. Kim, "An Efficient Decoding of CAVLC in H.264/AVC VideoCoding Standard," IEEE Transactions on Consumer Electronics, Vol. 51, No. 3, 2005.

[4] H. Chang, C. Lin, and J. Guo, "A Novel Low-Cost High-Performance VLSI Architecture for MPEG-4 AVC/H.264 CAVLC Decoding," IEEE International Symposium on Circuits and Systems, Vol. 6, pp.6110 - 6113, 2005.

[5] E. Iain, "H.264 and MPEG-4 Video Copression," Wiley, 2003.

[6] J. Nikara, S. Vassiliadis, J. Takala, P. Liuha, "Multiple-Symbol Parallel Decoding for Variable Length Codes," IEEE Trans. on Very Large Scale Integration Systems, Vol. 12, Issue 7, pp.676 - 685, 2004.

저 자 소 개



여 동 훈(정회원)
2006년 한양대학교 전자컴퓨터공학부 학사.
2008년 한양대학교 메카트로닉스 공학과 석사.
<주관심분야 : 저전력 설계, H.264, 반도체 설계>

신 현 철(평생회원)
1978년 서울대학교 전자공학과 학사.
1980년 한국과학기술원 전기 및 전자공학과 석사.
1983년~1987년 U.C. Berkeley Ph.D
1983년~1987년 Fulbright scholarship
1987년~1989년 MTS, AT&T Bell Lab's, Murray Hill N.J., USA
1989년~현재 한양대학교 전자컴퓨터공학부 교수
1997년~2008년 현재 IDEC 한양대학교 지역센터 센터장
2008년~현재 ITRC MDM연구센터 센터장
<주관심분야: CAD&VLSI, 통신용 반도체 설계, 저전력 설계>