논문 2012-49SD-3-4

H.264/AVC를 위한 고성능 CAVLC 부호화기 하드웨어 설계

(Hardware Design of High Performance CAVLC Encoder)

이 양 복*, 류 광 기**

(Yangbok Lee and Kwangki Ryoo)

요 약

본 논문에서는 H.264/AVC 부호화기의 성능 향상을 위해 고성능 CAVLC 부호화기의 하드웨어 구조를 제안한다. 기존의 CAVLC 부호화기는 변환계수의 재정렬 과정이 포함되어 변환계수를 저장해야 할 버퍼와 버퍼제어를 위한 추가적인 사이클이 필요하므로 하드웨어 면적이 증가하고 불필요한 사이클이 수행된다. 제안한 CAVLC는 CAVLC의 파라미터 중에 Runbefore를 순방향 탐색기법으로 계산하고 그 외 파라미터들은 역방향 탐색기법으로 계산하여 변환계수의 재정렬 과정을 수행하지 않는다. 또한, 제안한 CAVLC 부호화기에 조기 종료 모드를 적용하고 2단 파이프라인 구조를 사용하여 CAVLC의 수행 사이클 수를 감소시켰다. 제안한 CAVLC 부호화기의 하드웨어 구조를 매그나칩 공정 0.18㎛ 셀 라이브러리로 합성한 결과, 최대동작 주파수는 125MHz이며 게이트 수는 17k이다. 제안한 CAVLC 부호화기의 하드웨어 구조를 H.264/AVC 표준 참조 소프트웨어 JM13.2에서 추출한 데이터를 이용하여 테스트한 결과, 16x16 매크로블록을 처리하는데 평균적으로 36.0사이클이 소요되어 기존의 CAVLC 부호화기보다 성능이 57.8% 향상됨을 확인하였다.

Abstract

This paper presents optimized searching technique to improve the performance of H.264/AVC. By using the proposed forward and backward searching algorithm, redundant cycles of latency for data reordering can be removed. Furthermore, in order to reduce the total number of execution cycles of CAVLC encoder, early termination mode and two stage pipelined architecture are proposed. The experimental result shows that the proposed architecture needs only 36.0 cycles on average for each 16x16 macroblock encoding. The proposed architecture improves the performance by 57.8% than that of previous designs. The proposed CAVLC encoder was implemented using Verilog HDL and synthesized with Magnachip 0.18㎛ standard cell library. The synthesis result shows that the gate count is about 17K with 125Mhz clock frequency.

Keywords : CAVLC, 엔트로피 코딩, H.264/AVC, 부호화기

Ⅰ. 서 론

최근 저장 매체와 네트워크의 급격한 발전으로 유선 또는 무선의 접속회선이 고속화되었으며 저장매체도 대용량화되었다. 그러나 문자, 음성, 사진, 동영상과 같은 멀티미디어 정보량 역시 보다 좋은 색상, 화질이 요구되어 사용되는 정보량 역시 시대에 따라 증가해 가고 있어 영상 데이터를 직접 전송하거나 저장하는데 많은 어려움이 있다. 따라서 영상 데이터를 효율적으로 처리하기 위해서는 영상 데이터의 압축이 필요하다[1].

H.264/AVC는 ITU-T의 비디오 코딩 전문가 그룹(Video Coding Experts Group, VCEG)과 ISO/IEC의 동화상 전문가 그룹(Moving Picture Experts Group, MPEG)이 공동으로 JVT(Joint Video Team)을 구성하고 표준화를 진행한 결과물이다. H.264/AVC는 기존의 영상 압축 표준과 비교했을 때 압축률이 2배 이상 향상되었다. 압축 성능의 향상을 위해 1/4화소를 사용한 움직임 보상과 정수 기반 DCT, 향상된 엔트로피 부호화,

* 학생회원, ** 정회원-교신저자, 한밭대학교 정보통신공학과
(Department of Information and Communication Engineering, Hanbat National University)

※ 본 연구는 한밭대학교의 2011년 교내학술연구비사업과 IDEC의 CAD Tool 지원사업의 연구결과임.
접수일자 2011년12월23일, 수정완료일: 2012년3월7일

디블록킹필터 등의 새로운 압축 방식이 도입되었다[2]. H.264/AVC는 고효율의 압축성능을 갖지만, 상대적으로 방대한 계산량을 요구하기 때문에 고성능 하드웨어 설계가 요구된다.

H.264/AVC의 엔트로피 부호화기인 CAVLC는 인접한 심볼 사이의 상관관계를 이용한 부호화 방식을 채택하여 부호화 효율을 향상시켰다. 기존의 CAVLC 부호화기[3-4]는 4x4블록에서 CAVLC의 파라미터를 계산하기 위해서 지그재그 스캐닝 순서의 역방향으로 변환계수를 재정렬하고 변환계수를 탐색한다. 변환계수의 재정렬 과정은 변환계수를 저장해야 할 버퍼와 버퍼제어를 위한 추가적인 사이클이 필요하므로 CAVLC 부호화기의 하드웨어 면적이 증가되고 불필요한 사이클이 수행된다. 이와 같은 문제점을 해결하기 위해서 Hsia[5]는 순방향 탐색기법을 제안했다. 그러나 순방향 탐색기법에서 Level은 탐색방향과 다르게 역방향으로 부호화되기 때문에 부호화를 수행하기 전에 Level을 재정렬해야 한다. 이러한 문제점 때문에 Level은 부호화가 시작되고 2사이클 이후에 부호화가 가능하다.

본 논문에서는 기존의 CAVLC 부호화기의 문제점을 해결하기 위해서 변환계수의 재정렬 과정이 필요 없는 탐색기법을 제안한다. 제안한 탐색기법은 Level을 역방향 탐색기법으로 계산하고 그 외 파라미터들은 순방향 탐색기법으로 계산하여 변환계수의 재정렬 과정을 수행하지 않는다. 또한, 제안한 CAVLC 부호화기에 조기 종료 모드를 적용하고 2단 파이프라인 구조를 사용하여 CAVLC의 수행 사이클 수를 감소시켰다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 CAVLC의 알고리즘에 대하여 기술하며, Ⅲ장에서는 제안하는 CAVLC의 하드웨어 구조를 제시한다. Ⅳ장에서는 제안한 구조와 기존 구조를 비교 분석한다. 마지막으로 Ⅴ장에서는 결론을 도출한다.

Ⅱ. CAVLC 알고리즘

CAVLC는 지그재그 스캐닝 순서로 변환계수를 정렬하고 정렬된 변환계수의 역순에 따라 파라미터를 계산한다. 파라미터는 H.264/AVC 표준에 정의된 테이블을 이용하여 각각에 해당하는 코드워드로 생성된다. CAVLC는 5가지 순서로 변환계수를 다음과 같이 부호화한다.

1. 블록의 변환계수의 개수(CoeffToken)를 부호화한다. CoeffToken은 0이 아닌 변환계수의 개수(TotalCoeff)와 ±1(TrailingOnes)의 개수를 하나의 코드워드로 부호화된다. CoeffToken을 부호화 할 때 5가지의 CoeffToken 룩업 테이블 중 하나를 선택하여 부호화한다. 룩업 테이블의 선택은 현재 블록의 왼쪽 블록과 위쪽 블록의 TotalCoeff 평균값에 따라 이루어진다.
2. 변환계수 중에서 ±1인 변환계수(TrailingOnes)를 부호화 한다. 변환계수가 +1일 때에는 0으로, -1일 때에는 1로 부호화한다.
3. TrailingOnes를 제외한 0이 아닌 변환계수(Level)을 부호화한다. Level을 부호화 할 때 각 Level의 절대값이 일정치 이상 초과할 때마다 다음 부호화될 Level은 Suffix가 길은 룩업 테이블을 선택한다.
4. 0이 아닌 변환계수 앞에 존재하는 모든 0의 개수(Total_Zeros)를 부호화한다. TotalZeros를 부호화 할 때 TotalCoeff에 따라 룩업 테이블을 선택한다.
5. 0이 아닌 변환계수 앞에 존재하는 연속된 0의 개수(RunBefore)를 부호화한다. RunBefore를 부호화 할 때 부호화되지 않은 0의 개수(ZerosLeft)에 따라 룩업 테이블을 선택한다.

Ⅲ. 제안하는 CAVLC

본 논문에서 제안하는 탐색기법은 RunBefore를 순방향 탐색기법으로 계산하고 그 외 파라미터들은 역방향 탐색기법으로 계산한다. 또한, 제안하는 CAVLC 부호화기는 2단 파이프라인을 적용하였으며 파라미터를 탐색하는 과정에서는 한 사이클에 4개의 변환 계수를 탐색하고 파라미터를 코드워드로 생성하는 과정에서는 RunBefore와 Level을 2개씩 처리한다. 그리고 조기종료 모드를 적용하여 4x4블록의 변환 계수가 전부 0인 블록에 대해서는 1 사이클만에 종료된다.

제안한 CAVLC의 하드웨어 구조는 그림 1과 같이 순방향 제어기와 역방향 제어기, 상태 생성기, TotalCoeff 카운트, TrailingOne 카운트, TotalZeros 카운트, NonZeroCoeff 정렬기, Level 검출기, ZeroLeft 카운트, RunBefore 카운트, CoeffToken 테이블, Level 테이블, TotalCoeff 테이블, RunBefore 테이블, Packer로 구성된다.

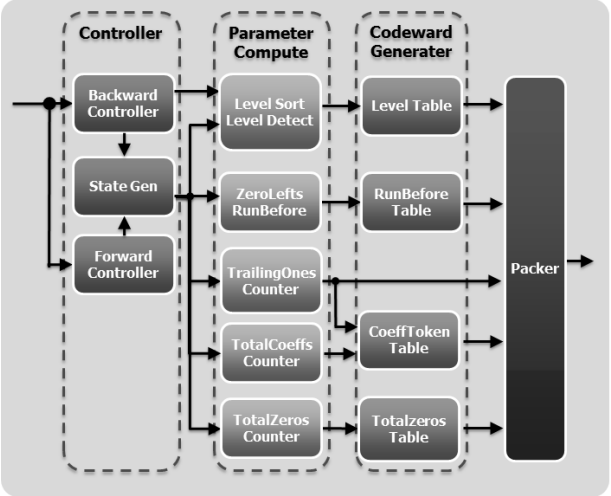

그림 1. 제안하는 CAVLC 부호화기 구조
Fig. 1. Architecture of the Proposed CAVLC encoder.

1. 순방향 제어기와 역방향 제어기

순방향 제어기와 역방향 제어기는 각 카운트에서 파라미터를 계산할 수 있도록 한 사이클에 4개씩 변환계수를 출력한다. 순방향 제어기는 지그재그 스캐닝 순서로 4사이클 동안 16개의 변환계수(FCoeff)를 상태 생성기로 전달한다. 그와 반대로 역방향 제어기는 지그재그 스캐닝 순서의 역순으로 16개의 변환계수(BCoeff)를 상태 생성기로 전달한다.

2. 상태 생성기

상태 생성기는 변환계수의 상태를 생성한다. 식 (1)과 같이 BCoeff1 ~ BCoeff4와 FCoeff1 ~ FCoeff4의 상태에 따라 S1 ~ S12를 생성하여 각 카운트에 전달한다.

$$\begin{array}{l}
\text{if } (BCoeff1 \neq 0) \quad then \quad S1=1, \quad else \quad S1=0 \\
\text{if } (BCoeff2 \neq 0) \quad then \quad S2=1, \quad else \quad S2=0 \\
\text{if } (BCoeff3 \neq 0) \quad then \quad S3=1, \quad else \quad S3=0 \\
\text{if } (BCoeff4 \neq 0) \quad then \quad S4=1, \quad else \quad S4=0 \\
\\
\text{if } (BCoeff1=\pm 1) \quad then \quad S5=1, \quad else \quad S5=0 \\
\text{if } (BCoeff2=\pm 1) \quad then \quad S6=1, \quad else \quad S6=0 \\
\text{if } (BCoeff3=\pm 1) \quad then \quad S7=1, \quad else \quad S7=0 \\
\text{if } (BCoeff4=\pm 1) \quad then \quad S8=1, \quad else \quad S8=0 \\
\\
\text{if } (FCoeff1 \neq 0) \quad then \quad S9=1, \quad else \quad S9=0 \\
\text{if } (FCoeff2 \neq 0) \quad then \quad S10=1, \quad else \quad S10=0 \\
\text{if } (FCoeff3 \neq 0) \quad then \quad S11=1, \quad else \quad S11=0 \\
\text{if } (FCoeff4 \neq 0) \quad then \quad S12=1, \quad else \quad S12=0
\end{array} \tag{1}$$

S1~S12의 스케줄은 표 1과 같다. S1~S4와 S9~S12는 변환 계수가 0이 아닌 경우에 1이 되고 S5~S8는 변환 계수가 ±1인 경우에 1이 된다.

표 1. S1 ~ S12 스케줄
Table 1. S1 ~ S12 schedule.

CycleCnt	0	1	2	3
Bcoeff1-4	0 0 0 0	1 1 0 0	0 2 8 0	0 2 6 9
Fcoeff1-4	9 6 2 0	0 8 2 0	0 0 1 1	0 0 0 0
S1 ~ S4	0 0 0 0	1 1 0 0	0 1 1 0	0 1 1 1
S5 ~ S8	0 0 0 0	1 1 0 0	0 0 0 0	0 0 0 0
S9 ~ S12	1 1 1 0	0 1 1 0	0 0 1 1	0 0 0 0

3. TotalCoeff 카운트

TotalCoeff 카운트는 0이 아닌 변환 계수의 개수를 카운트한다. TotalCoeff는 식 (2)와 같이 S1 ~ S4를 이용하여 카운트된다.

$$\begin{array}{l}
case(S1, S2, S3, S4) \\
0000: TotalCoeff = TotalCoeffPre \\
0001: TotalCoeff = TotalCoeffPre + 1 \\
0010: TotalCoeff = TotalCoeffPre + 1 \\
0011: TotalCoeff = TotalCoeffPre + 2 \\
0100: TotalCoeff = TotalCoeffPre + 1 \\
0101: TotalCoeff = TotalCoeffPre + 2 \\
0110: TotalCoeff = TotalCoeffPre + 2 \\
0111: TotalCoeff = TotalCoeffPre + 3 \\
1000: TotalCoeff = TotalCoeffPre + 1 \\
1001: TotalCoeff = TotalCoeffPre + 2 \\
1010: TotalCoeff = TotalCoeffPre + 2 \\
1011: TotalCoeff = TotalCoeffPre + 3 \\
1100: TotalCoeff = TotalCoeffPre + 2 \\
1101: TotalCoeff = TotalCoeffPre + 3 \\
1110: TotalCoeff = TotalCoeffPre + 3 \\
1111: TotalCoeff = TotalCoeffPre + 4
\end{array} \tag{2}$$

식 (2)에서 TotalCoeff는 0이 아닌 변환 계수의 개수를 나타내며 TotalCoeffPre는 이전 사이클에서 TotalCoeff를 나타낸다. TotalCoeff의 스케줄은 표 2와 같다. TotalCoeff는 S1~S4 중에서 값이 1인 상태의 개수만큼 누적된다. 즉, S1~S4가 0001, 0010 0100, 1000일 경우에 1이 누적되고 S1~S4가 0011, 0101, 0110, 1001, 1010, 1100인 경우에 2가 누적된다. 마찬가지로 S1~S4가 0111, 1011, 1101, 1110인 경우에는 3이 누적되며 S1~S4가 1111인 경우에는 4가 누적된다.

표 2. TotalCoeff 스케줄
Table 2. TotalCoeff schedule.

CycleCnt	0	1	2	3
Bcoeff1-4	0 0 0 0	1 1 0 0	0 2 8 0	0 2 6 9
S1 ~ S4	0 0 0 0	1 1 0 0	0 1 1 0	0 1 1 1
TotalCoeff	0	2	4	7
TotalCoeffPre	0	0	2	4

4. TrailingOnes 카운트

TrailingOnes 카운트는 TrailingOnes의 개수를 카운트한다. TrailingOnes을 카운트하기 위해서는 변환 계수가 Level인지 TrailingOnes인지 구분해야하므로 LLF(Last Level Flag)와 TrailingOnes의 개수(Tnum)를 이용하여 TrailingOnes를 카운트한다. LLF는 식 (3)과 같이 S1 ~ S8을 이용하여 판단한다.

$$
\begin{aligned}
&IsLevel1 = S1 \wedge (\overline{S5})\\
&IsLevel2 = S2 \wedge (\overline{S6})\\
&IsLevel3 = S3 \wedge (\overline{S7})\\
&IsLevel4 = S4 \wedge (\overline{S8})\\
&case(IsLevel1, IsLevel2, IsLevel3, IsLevel4)\\
&\quad 0000 : LLF = 0000\\
&\quad 0001 : LLF = 0001\\
&\quad 0010 : LLF = 0011\\
&\quad 0011 : LLF = 0011\\
&\quad 0100 : LLF = 0111\\
&\quad 0101 : LLF = 0111\\
&\quad 0110 : LLF = 0111\\
&\quad 0111 : LLF = 0111\\
&\quad 1000 : LLF = 1111\\
&\quad 1001 : LLF = 1111\\
&\quad 1010 : LLF = 1111\\
&\quad 1011 : LLF = 1111\\
&\quad 1100 : LLF = 1111\\
&\quad 1101 : LLF = 1111\\
&\quad 1110 : LLF = 1111\\
&\quad 1111 : LLF = 1111\\
&\mathrm{if}\ (LLFPre \neq 0000)\ then\\
&\quad LLF = 1111
\end{aligned}
\tag{3}
$$

식 (3)에서 $\wedge$는 비트 논리 연산자 And를 나타내며 Islevel1 ~ Islevel4는 0과 ±1이 아닌 변환 계수인 경우에 1이 된다. 그리고 LLFPre는 이전사이클의 LLF를 나타내며, LLF는 현재 사이클까지 Level이 발견되었는지를 나타낸다. LLF 스케줄은 표 3와 같다. 첫 번째 사이클에서 IsLevel1 ~ Islevel4가 0000이기 때문에 LFF는 0000이 되며 두 번째 사이클에서는 IsLevel1 ~ Islevel4가 1100이기 때문에 LLF는 1111이 된다.

표 3. LLF 스케줄
Table 3. LLF schedule.

CycleCnt	0	1	2	3
Bcoeff1 ~ Bcoeff4	0 0 0 0	1 1 0 0	0 2 8 0	0 2 6 9
S1 ~ S4	0 0 0 0	1 1 0 0	0 1 1 0	0 1 1 1
S5 ~ S8	0 0 0 0	1 1 0 0	0 0 0 0	0 0 0 0
IsLevel1 ~ IsLevel4	0 0 0 0	0 0 0 0	0 1 1 0	0 1 1 1
LLF	0 0 0 0	0 0 0 0	0 1 1 1	1 1 1 1
LLFPre	0 0 0 0	0 0 0 0	0 0 0 0	0 1 1 1

TrailingsOnes의 개수는 식 (4)와 같이 LLF와 S5 ~ S8을 이용하여 카운트된다.

$$
\begin{aligned}
&IsAbsone = S5, S6, S7, S8\\
&IsTNumFlag = IsAbsone \wedge \overline{LLF}\\
&case(IsTnumFlag)\\
&\quad 0000 : TNum = TNumPre + 0\\
&\quad 0001 : TNum = TNumPre + 1\\
&\quad 0010 : TNum = TNumPre + 1\\
&\quad 0011 : TNum = TNumPre + 2\\
&\quad 0100 : TNum = TNumPre + 1\\
&\quad 0101 : TNum = TNumPre + 2\\
&\quad 0110 : TNum = TNumPre + 2\\
&\quad 0111 : TNum = TNumPre + 3\\
&\quad 1000 : TNum = TNumPre + 1\\
&\quad 1001 : TNum = TNumPre + 2\\
&\quad 1010 : TNum = TNumPre + 2\\
&\quad 1011 : TNum = TNumPre + 3\\
&\quad 1100 : TNum = TNumPre + 2\\
&\quad 1101 : TNum = TNumPre + 3\\
&\quad 1110 : TNum = TNumPre + 3\\
&\quad 1111 : TNum = TNumPre + 3\\
&\mathrm{if}\ (TNum > 3)\ then\\
&\quad TNum = 3
\end{aligned}
\tag{4}
$$

식 (4)에서 IsTNumFlag는 TrailingOnes가 검출된 경우에 각 인덱스는 1이 되며 TNumPre는 이전 사이클에서 TNum을 나타낸다. TNum 스케줄은 표 4와 같다. 4개의 변환 계수 중 3개의 변환 계수가 TrailingOnes일 때는 3이 누적된다. IsTNumFlag가 0111, 1011, 1101, 1110, 1111인 경우가 해당한다. 4개의 변환 계수 중 2개의 변환 계수가 TrailingOnes일 때는 2가 누적된다. IsTNumFlag가 0011, 0101, 0110, 1001, 1010, 1100인 경우가 해당한다. 4개의 변환 계수 중 1개의 변환 계수가 TrailingOnes일 때는 1이 누적된다. IsTNumFlag가 0001, 0010, 0100, 1000인 경우가 해당한다.

표 4. TNum 스케줄
Table 4. TNum schedule.

CycleCnt	0	1	2	3
Bcoeff1 ~ Bcoeff4	0 0 0 0	1 1 0 0	0 2 8 0	0 2 6 9
S5 ~ S8	0 0 0 0	1 1 0 0	0 1 1 0	0 1 1 1
LLF	0 0 0 0	0 0 0 0	0 1 1 1	1 1 1 1
IsTNumFlag	0 0 0 0	1 1 0 0	0 0 0 0	0 0 0 0
TNum	0	2	2	2
TNumPre	0	0	2	2

5. TotalZeros 카운트

TotalZeros 카운트는 0인 변환 계수의 개수를 카운트한다. 0인 변환 계수의 개수는 식 (5)와 같이 S1 ~ S4를 이용하여 카운트된다.

$$
\begin{aligned}
&case(S1, S2, S3, S4)\\
&\quad 0000 : TotalZeros = LCPre\ ?\ TotalZerosPre + 4 : 0\\
&\quad 0001 : TotalZeros = LCPre\ ?\ TotalZerosPre + 3 : 0\\
&\quad 0010 : TotalZeros = LCPre\ ?\ TotalZerosPre + 3 : 1\\
&\quad 0011 : TotalZeros = LCPre\ ?\ TotalZerosPre + 2 : 0\\
&\quad 0011 : TotalZeros = LCPre\ ?\ TotalZerosPre + 2 : 0\\
&\quad 0100 : TotalZeros = LCPre\ ?\ TotalZerosPre + 3 : 2\\
&\quad 0101 : TotalZeros = LCPre\ ?\ TotalZerosPre + 2 : 1\\
&\quad 0110 : TotalZeros = LCPre\ ?\ TotalZerosPre + 2 : 1\\
&\quad 0111 : TotalZeros = LCPre\ ?\ TotalZerosPre + 1 : 0\\
&\quad 1000 : TotalZeros = LCPre\ ?\ TotalZerosPre + 3 : 3\\
&\quad 1001 : TotalZeros = LCPre\ ?\ TotalZerosPre + 2 : 2\\
&\quad 1010 : TotalZeros = LCPre\ ?\ TotalZerosPre + 2 : 2\\
&\quad 1011 : TotalZeros = LCPre\ ?\ TotalZerosPre + 1 : 1\\
&\quad 1100 : TotalZeros = LCPre\ ?\ TotalZerosPre + 2 : 2\\
&\quad 1101 : TotalZeros = LCPre\ ?\ TotalZerosPre + 1 : 1\\
&\quad 1110 : TotalZeros = LCPre\ ?\ TotalZerosPre + 1 : 1\\
&\quad 1111 : TotalZeros = LCPre\ ?\ TotalZerosPre : 0\\
&\mathrm{if}((LCPre = 1) \wedge (S1, S2, S3, S4 = 0000))\ then\\
&\quad LC = 1\\
&else\\
&\quad LC = 0
\end{aligned}
\tag{5}
$$

식 (5)에서 TotalZerosPre는 이전 사이클의 TotalZeros를 나타내며 TotalZeros는 현재 사이클까지 0의 개수를 나타낸다. LCPre는 이전 사이클의 LC를 나타내며 LC는 지그재그 스캐닝 순서로 정렬된 변환 계수에서 마지막에 있는 0이 아닌 변환 계수가 발견되었는지를 나타낸다. TotalZeros의 스케줄은 표 5과 같다. LC는 이전 사이클에서 0이 아닌 변환 계수가 발견되거나 현재 사이클에서 0이 아닌 변환 계수가 발견된 경우에 1이 된다. TotalZeros는 지그재그 스캐닝 순서로 정렬된 변환 계수에서 마지막에 있는 0이 아닌 변환 계수 앞에 있는 0을 카운트하기 때문에 LCPre에 따라 누적되는 값이 다르다. 이전 사이클에서 0이 아닌 변환 계수가 발견되었을 경우 입력된 변환 계수에서 0의 개수만큼 TotalZeros에 누적된다. LCPre가 1인 경우가 해당한다. LCPre가 0인 경우에는 입력된 변환 계수 중 0이 아닌 변환 계수가 있을 경우 0이 아닌 변환 계수 앞에 있는 0의 개수만큼 TotalZeros에 누적된다.

표 5. TotalZeros 스케줄
Table 5. TotalZeros schedule.

CycleCnt	0	1	2	3
Bcoeff1 ~ Bcoeff4	0 0 0 0	1 1 0 0	0 2 8 0	0 2 6 9
S1 ~ S4	0 0 0 0	1 1 0 0	0 1 1 0	0 1 1 1
LC	0	1	1	1
LCpre	0	0	1	1
TotalZeros	0	2	4	5
TotalZerosPre	0	0	2	4

6. RunBefore, ZeroLeft 카운트

RunBefore, ZeroLeft 카운트는 Runbefore와 ZeroLeft를 카운트한다. 그 중에 ZeroLeft는 변환 계수 앞에 존재하는 0의 개수 중 RunBefore로 부호화 되지 않은 0의 개수를 나타내며 ZeroLeft는 식 (6)과 같이 S9 ~ S12를 이용하여 카운트한다.

$$
\begin{aligned}
&FstLeft = FthLeft + ZerosTempPre\\
&case(S9, S10, S11, S12)\\
&\quad 0000 : SndLeft = FstLeft + 1,\ TrdLeft = SndLeft + 1,\\
&\qquad FthLeft = TrdLeft + 1,\ ZerosTemp = 1\\
&\quad 0001 : SndLeft = FstLeft + 1,\ TrdLeft = SndLeft + 1,\\
&\qquad FthLeft = TrdLeft + 1,\ ZerosTemp = 0\\
&\quad 0010 : SndLeft = FstLeft + 1,\ TrdLeft = SndLeft + 1,\\
&\qquad FthLeft = TrdLeft,\ ZerosTemp = 1\\
&\quad 0011 : SndLeft = FstLeft + 1,\ TrdLeft = SndLeft + 1,\\
&\qquad FthLeft = TrdLeft,\ ZerosTemp = 0\\
&\quad 0100 : SndLeft = FstLeft + 1,\ TrdLeft = SndLeft,\\
&\qquad FthLeft = TrdLeft + 1,\ ZerosTemp = 1\\
&\quad 0101 : SndLeft = FstLeft + 1,\ TrdLeft = SndLeft,\\
&\qquad FthLeft = TrdLeft + 1,\ ZerosTemp = 0\\
&\quad 0110 : SndLeft = FstLeft + 1,\ TrdLeft = SndLeft,\\
&\qquad FthLeft = TrdLeft,\ ZerosTemp = 1\\
&\quad 0111 : SndLeft = FstLeft + 1,\ TrdLeft = SndLeft,\\
&\qquad FthLeft = TrdLeft,\ ZerosTemp = 0\\
&\quad 1000 : SndLeft = FstLeft,\ TrdLeft = SndLeft + 1,\\
&\qquad FthLeft = TrdLeft + 1,\ ZerosTemp = 1\\
&\quad 1001 : SndLeft = FstLeft,\ TrdLeft = SndLeft + 1,\\
&\qquad FthLeft = TrdLeft + 1,\ ZerosTemp = 0\\
&\quad 1010 : SndLeft = FstLeft,\ TrdLeft = SndLeft + 1,\\
&\qquad FthLeft = TrdLeft,\ ZerosTemp = 1\\
&\quad 1011 : SndLeft = FstLeft,\ TrdLeft = SndLeft + 1,\\
&\qquad FthLeft = TrdLeft,\ ZerosTemp = 0\\
&\quad 1100 : SndLeft = FstLeft,\ TrdLeft = SndLeft,\\
&\qquad FthLeft = TrdLeft + 1,\ ZerosTemp = 1,\\
&\quad 1101 : SndLeft = FstLeft,\ TrdLeft = SndLeft,\\
&\qquad FthLeft = TrdLeft + 1,\ ZerosTemp = 0,\\
&\quad 1110 : SndLeft = FstLeft,\ TrdLeft = SndLeft,\\
&\qquad FthLeft = TrdLeft,\ ZerosTemp = 1,\\
&\quad 1111 : SndLeft = FstLeft,\ TrdLeft = SndLeft,\\
&\qquad FthLeft = TrdLeft,\ ZerosTemp = 0,
\end{aligned}
\tag{6}
$$

식 (6)에서 ZerosTemp는 네 번째 변환 계수의 상태를 나타내며 FstLeft은 첫 번째 변환 계수의 ZeroLeft를 나타내고 SndLeft은 두 번째 변환 계수의 ZeroLeft를 나타낸다. 또한, TrdtLeft은 세 번째 변환 계수의 ZeroLeft를 나타내고 FthLeft은 네 번째 변환 계수의 ZeroLeft를 나타낸다. ZerosTempPre와 FthLeftPre는 이전 사이클의 ZerosTemp와 FthLeft를 나타낸다. ZeroLeft의 스케줄은 표 6과 같다. 첫 번째 사이클에서 S9~S12가 0001이기 때문에 FstLeft, SndLeft, TrdLeft,

표 6. ZeroLeft 스케줄
Table 6. ZeroLeft schedule.

CycleCnt	0	1	2	3
Fcoeff1 ~ Fcoeff4	9 6 2 0	0 8 2 0	0 0 1 1	0 0 0 0
S9 ~ S12	1 1 1 0	0 1 1 0	0 0 1 1	0 0 0 0
ZeroLeft	0 0 0 0	1 2 2 2	3 4 5 5	5 5 5 5
ZerosTemp	1	1	0	0

FthLeft는 현재 값을 유지하고 ZerosTemp는 1이 된다. 두 번째 사이클에서는 FstLeft는 이전 사이클에서 ZerosTemp가 1이기 때문에 1이 누적된다. 또한, S9 ~ S12가 0110이기 때문에 SndLeft에서 1이 누적되고 ZerosTemp는 1이 된다. 세 번째 사이클에서는 FstLeft는 이전 사이클에서 ZerosTemp가 1이기 때문에 1이 누적된다. 또한, S9 ~ S12가 0011이기 때문에 SndLeft와 TrdLeft에서 1이 누적되고 ZerosTemp는 0이 된다. 네 번째 사이클에서는 FstLeft는 이전 사이클에서 ZerosTemp가 0이기 때문에 값이 유지된다. 또한, S9 ~S12가 0000이기 때문에 SndLeft, TrdLeft, FthLeft는 값을 유지하고 ZerosTemp는 0이 된다.

RunBefore는 변환 계수 앞에 존재하는 연속적인 0의 개수를 나타내며 RunBefore는 식 (7)과 같이 S9 ~ S12를 이용하여 카운트한다.

$$
\begin{aligned}
&State = S9, S10, S11, S12 \\
&\text{if}\,(StatePre[0] = 1)\ then \\
&\quad FstRun = 0 \\
&else \\
&\quad FstRun = FthRunPre + ZerosTempPre \\
&case(State) \\
&\quad 0000: SndRun = FstRun + 1,\ TrdRun = SndRun + 1, \\
&\qquad FthRun = TrdRun + 1,\ ZerosTemp = 1 \\
&\quad 0001: SndRun = FstRun + 1,\ TrdRun = SndRun + 1, \\
&\qquad FthRun = TrdRun + 1,\ ZerosTemp = 0 \\
&\quad 0010: SndRun = FstRun + 1,\ TrdRun = SndRun + 1, \\
&\qquad FthRun = 0,\ ZerosTemp = 1 \\
&\quad 0011: SndRun = FstRun + 1,\ TrdRun = SndRun + 1, \\
&\qquad FthRun = 0,\ ZerosTemp = 0 \\
&\quad 0100: SndRun = FstRun + 1,\ TrdRun = 0, \\
&\qquad FthRun = TrdRun + 1,\ ZerosTemp = 1 \\
&\quad 0101: SndRun = FstRun + 1,\ TrdRun = 0, \\
&\qquad FthRun = TrdRun + 1,\ ZerosTemp = 0 \\
&\quad 0110: SndRun = FstRun + 1,\ TrdRun = 0, \\
&\qquad FthRun = 0,\ ZerosTemp = 1 \\
&\quad 0111: SndRun = FstRun + 1,\ TrdRun = 0, \\
&\qquad FthRun = 0,\ ZerosTemp = 0 \\
&\quad 1000: SndRun = 0,\ TrdRun = SndRun + 1, \\
&\qquad FthRun = TrdRun + 1,\ ZerosTemp = 1 \\
&\quad 1001: SndRun = 0,\ TrdRun = SndRun + 1, \\
&\qquad FthRun = TrdRun + 1,\ ZerosTemp = 0 \\
&\quad 1010: SndRun = 0,\ TrdRun = SndRun + 1, \\
&\qquad FthRun = 0,\ ZerosTemp = 1 \\
&\quad 1011: SndRun = 0,\ TrdRun = SndRun + 1, \\
&\qquad FthRun = 0,\ ZerosTemp = 0 \\
&\quad 1100: SndRun = 0,\ TrdRun = 0, \\
&\qquad FthRun = TrdRun + 1,\ ZerosTemp = 1 \\
&\quad 1101: SndRun = 0,\ TrdRun = 0, \\
&\qquad FthRun = 0,\ ZerosTemp = 0 \\
&\quad 1110: SndRun = 0,\ TrdRun = 0, \\
&\qquad FthRun = 0,\ ZerosTemp = 1 \\
&\quad 1111: SndRun = 0,\ TrdRun = 0, \\
&\qquad FthRun = 0,\ ZerosTemp = 0
\end{aligned}
\tag{7}
$$

식 (7)에서 ZerosTemp는 네 번째 변환 계수의 상태를 나타내며 FstRun은 첫 번째 변환 계수의 RunBefore를 나타내고 SndRun은 두 번째 변환 계수의 RunBefore를 나타낸다. 또한, TrdtRun은 세 번째 변환 계수의 RunBefore를 나타내고 FthRun은 네 번째 변환 계수의 RunBefore를 나타낸다. ZerosTempPre와 FthRunPre, StatePre는 이전 사이클의 ZerosTemp와 FthRun, State를 나타낸다. RunBefore의 스케줄은 표 7과 같다. 첫 번째 사이클에서 S9~S12가 0001이기 때문에 FstRun, SndRun, TrdRun, FthRun는 0이 되고 ZerosTemp는 1이 된다. 두 번째 사이클에서는 FstRun는 이전 사이클에서 ZerosTemp가 1이기 때문에 1이 누적된다. 또한, S9~S12가 0110이기 때문에 SndRun에서 1이 누적되며 TrdRun, FthRun는 0이 되고 ZerosTemp는 1이 된다. 세 번째 사이클에서는 FstRun는 이전 사이클에서 ZerosTemp가 1이기 때문에 1이 누적된다. 또한, S9 ~ S12가 0011이기 때문에 SndRun와 TrdRun에서 1이 누적되고 FthRun, ZerosTemp는 0이 된다. 네 번째 사이클에서는 FstRun는 StatePre[0]이 1이기 때문에 0이 된다. 또한, S9 ~ S12가 0000이기 때문에 SndRun, TrdRun, FthRun, ZerosTemp는 0이 된다.

표 7. RunBefore 스케줄
Table 7. RunBefore schedule.

CycleCnt	0	1	2	3
Fcoeff1 ~ Fcoeff4	9 6 2 0	0 8 2 0	0 0 1 1	0 0 0 0
S9 ~ S12	1 1 1 0	0 1 1 0	0 0 1 1	0 0 0 0
RunBefore	0 0 0 0	1 2 0 0	1 2 3 0	0 0 0 0
ZerosTemp	1	1	0	0

7. 코드워드 생성기

코드워드 생성기는 5가지 구문을 H.264/AVC 표준에 정의된 테이블을 이용하여 각각에 해당하는 코드워드로 부호화한다. 그림 2와 같이 코드워드 생성기는 각 파라미터를 병렬로 부호화한다. Level은 검출 순서와 부호화 순서가 동일하기 때문에 변환 계수의 재정렬 과정이

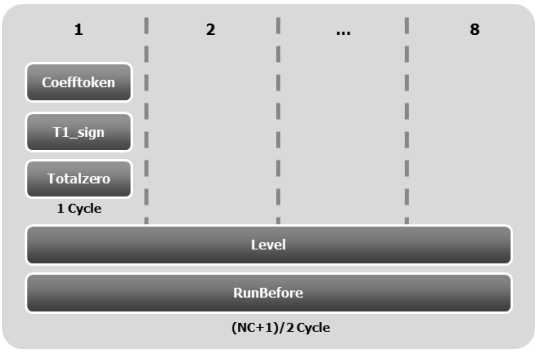

그림 2. CAVLC 부호화 스케줄
Fig. 2. CAVLC coding schedule.

필요 없이 첫 번째 사이클부터 부호화된다. RunBefore와 Level은 1 사이클에 2개씩 (NC+1)/2 사이클 동안 부호화된다.

8. 파이프라인

제안하는 CAVLC 부호화기는 그림 3과 같이 2단 파이프라인을 채택했다. 첫 번째 단계는 4사이클 동안 TotalCoeff, TrailingOnes, TotalZeros을 계산한다. 두 번째 단계는 ZeroLeft, RunBefore, Level를 계산하며 각 파라미터를 테이블을 이용하여 코드워드로 부호화하고 각 코드워드를 하나의 비트스트림으로 생성한다.

변환 계수가 0으로만 이루어진 4x4블록은 파라미터를 계산할 필요가 없기 때문에 제안하는 CAVLC 부호화기는 TotalCoeff가 0인 4x4블록이 연속적으로 2번 들어오는 경우에는 조기 종료되어 1사이클 만에 수행된다.

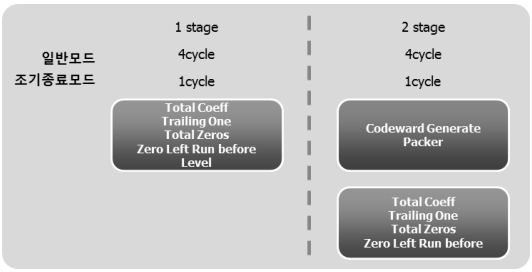

그림 3. 파이프라인 스케줄
Fig. 3. Pipeline schedule.

Ⅳ. 실 험 및 고찰

본 논문에서는 CIF영상(352화소 x 288화소, 300프레임, 4:2:0)을 이용하여 H.264/AVC 표준 참조 소프트웨어 JM13.2[6]에서 비트스트림을 추출하였고, 제안한 CAVLC 부호화기를 시뮬레이션 한 결과와 참조 소프트웨어에서 추출한 비트스트림을 비교하여 정상적으로 부호화됨을 확인하였다.

표 8. CALVC 부호화기 하드웨어 비교
Table 8. Comparison of the CAVLC encoder architectures.

	Tsai[7]	우정욱[8]	Hsia[5]	Proposed
Technology	0.18um	0.18um	0.18um	0.18um
Frequency	125MHz	81MHz	125MHz	125MHz
Gate Counter	10.5k	16.4k	15k	17k

표 9. 매크로블록 당 평균 수행 사이클 수
Table 9. Cycles/MB.

QP	Sequence	Tsai[7]	Hsia[5]	Proposed	Reduced
10	akiyo	100.8	89.8	39.1	56.4%
	coastguard	355.3	266.2	105.5	60.4%
	hall	374.8	256.3	102.1	60.2%
	mobile	375	294.5	118.8	59.6%
	mother	339.2	200.8	91.1	54.6%
	silent	172.3	198.2	86.3	56.4%
20	akiyo	31.4	18.2	8.0	55.9%
	coastguard	218.3	135.0	60.2	55.4%
	hall	207.7	158.5	68.4	56.8%
	mobile	295.5	177.5	75.3	57.6%
	mother	52.4	33.7	15.0	55.6%
	silent	172.3	43.5	19.4	55.5%
30	akiyo	6.3	3.2	1.3	58.5%
	coastguard	91.6	51.1	23.5	53.9%
	hall	13.8	8.7	3.7	57.3%
	mobile	120.3	68.6	28.4	58.7%
	mother	10.8	5.9	2.5	57.8%
	silent	21.4	11.7	5.0	57.3%
40	akiyo	1.7	0.5	0.2	61.5%
	coastguard	9.3	8.8	3.6	59.7%
	hall	3.7	1.5	0.6	59.4%
	mobile	20.7	12.5	4.6	63.4%
	mother	1.9	1.0	0.4	62.5%
	silent	4.3	2.7	1.0	61.9%
average		125.0	85.4	36.0	57.8%

표 8은 제안한 CAVLC 부호화기를 매그나칩 공정 0.18㎛ 셀 라이브러리로 Design Compiler에서 합성한 결과이다.

Hsia[5]는 파라미터를 계산하는 과정에서 한 사이클에 2개의 변환계수를 처리하고 RunBefore 테이블과 Level 테이블이 1개씩 존재하였으며 제안한 CAVLC 부호화기는 파라미터를 계산하는 과정에서 한 사이클에 4개의 변환계수를 처리하고 RunBefore 테이블과 Level테이블

이 2개씩 존재하여 Hsia[5]보다 2k증가하였으며 최대동작 주파수는 125Mhz로 동일하다.

표 9는 기존의 CAVLC 부호화기와 성능비교를 위해 16x16 매크로블록 당 평균 수행 사이클 수를 측정한 결과이다.

제안한 CAVLC 부호화기는 Hsia[5]보다 최저 53.9%에서 최고 63.4%까지 성능이 향상됨을 알 수 있다. 또한, 제안한 구조의 수행 사이클 수는 평균 36.0사이클이며 Hsia[5]보다 평균 57.8%가 향상되었다. 성능의 향상은 QP가 40일 때 가장 컸으며, QP가 10일 때 미비하였다. 이는 QP가 클수록 Level 부호의 횟수가 적어지고 TotalCoeff가 0인 4x4블록이 연속적으로 들어오는 경우가 많아지기 때문이다.

V. 결 론

본 논문에서는 CAVLC 부호화기의 성능을 향상시키기 위한 탐색기법과 효율적인 파이프라인 구조를 제안하고 설계 및 검증하였다. 제안한 구조는 RunBefore을 순방향 탐색기법으로 계산하고 그 외 파라미터는 역방향 탐색기법으로 계산하여 변환계수의 재정렬 과정이 필요 없다. 또한, 제안한 CAVLC 부호화기에 조기 종료 모드를 적용하고 2단 파이프라인 구조를 사용하여 수행 사이클 수를 감소시켰다. 제안한 CAVLC 부호화기를 매그나칩 공정 0.18㎛ 셀 라이브러리로 합성한 결과, 최대동작 주파수는 125MHz이고 게이트 수는 17k이다. 제안한 CAVLC의 하드웨어 구조를 H.264/AVC 표준 참조 소프트웨어 JM13.2에서 추출한 데이터를 이용하여 테스트한 결과, 16x16 매크로블록을 처리하는데 평균적으로 36.0사이클이 소요되어 기존의 CAVLC 부호화기보다 성능이 57.8% 향상됨을 확인하였다.

참 고 문 헌

[1] 정제창, *H.264/AVC 비디오 압축 표준, 홍릉과학출판사*, 2005.

[2] J. V. Team, *Advanced Video coding for generic audiovisual services*, ITU-T Recommendation H.264 and ISO/IEC 14496-10 AVC, May 2005.

[3] C. D. Chien, K. P. Lu, Y. H. Shih, and J. I. Guo, “A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications,” *in Proc. ISCAS*, pp. 3838 - 3841, May 2006.

[4] C. A. Rahman and W. Badawy, “CAVLC encoder design for real-time mobile video applications,” *IEEE Trans. Circuits Syst.* II, Exp. Briefs, vol. 54, no. 10, pp. 873 - 877, Oct. 2007.

[5] S. C. Hsia and W. H. Liao, “Forward Computations for Context-Adaptive Variable-Length Coding Design,” *IEEE Trans. Circuits Syst.* II, Exp. Briefs, vol. 57, no. 8, pp. 637-641, Aug. 2010.

[6] Joint Video Team(JVT) Reference Software JM 13.2.

[7] T. H. Tsai, S. P. Chang, T. L. Fang, “Highly efficient CAVLC encoder for MPEG-4 AVC/H.264,” *Circuits, Devices & Systems*, Volume 3, Issue 3, pp. 116-124, June 2009.

[8] 우정욱, 이원재, 김재석, “실시간 HD급 영상 처리를 위한 H.264/AVC CAVLC 부호화기의 하드웨어 구조 설계,” 전자공학회, 제 44 권, SD 편, 제 7 호, pp. 45-53, July 2007.

저 자 소 개

이 양 복(학생회원)

2010년 한밭대학교 공과대학 정보통신공학과 공학사
2012년 한밭대학교 정보통신전문대학원 정보통신공학과 공학석사
2012년～현재 ㈜실리콘웍스 개발본부 연구원

<주관심분야 : SoC 플랫폼 설계 및 검증, 영상신호처리>

류 광 기(정회원)

1986년 한양대학교 공과대학 전자공학과 공학사
1988년 한양대학교 대학원 전자공학과 공학석사
2000년 한양대학교 대학원 전자공학과 공학박사
1991년～1994년 육군사관학교 교수부 전자공학과 전임강사
2000년～2002년 한국전자통신연구원 시스템 IC 설계팀 선임연구원
2003년～현재 한밭대학교 정보통신공학과 부교수

<주관심분야 : SoC 플랫폼 설계 및 검증, 하드웨어/소프트웨어 통합설계 및 검증, 멀티미디어 코덱 설계>