H.264/AVC의 CAVLC 디코더를 위한 Coeff_Token 블록의 저면적 저전력 설계

정대진, 이강
한동대학교 전산전자공학부

Low Power and Low Area Degign of Coeff_token block for CAVLC decoder of H.264/AVC

Dae Jin Jeong and Kang Yi
School of Computer Science and Electrical Engineering,
Handong Global University

요 약

본 논문은, H,264/AVC 비디오 코덱의 저전력용 CAVLC 디코더를 위한 coeff_token 회로의 면적을 최적화 한 설계를 제시한다. CAVLC 디코더의 전력 소비를 줄이기 위해서 coeff_token 회로에서의 메모리 참조 빈도수를 줄이는 여러 가지 방법이 제안되어 왔다. 본 논문에서는 기존의 저전력용으로 개발된 coeff_token 회로 중 가장 전력 소비가 낮은 방식의 메모리 구조와 수식 계산 회로를 변형시켜서 전력 소비를 같은 수준으로 유지하면서도 면적을 더욱 줄이는 방법을 제안한다. 본 연구결과를 삼성 0.18 um 공정을 대상으로 합성한 결과 기존 방식에 비해서 1.1% 면적이 줄어드는 성과를 거두었다.

1. 서론

디지털 비디오 압축 기술의 최신 표준인 H.264가 높은 압축효율로 인해서 빠르게 주목을 받고 있다[1]. 이러한 멀티미디어 디지털 시스템에서 전력소비는 설계 시에 매우 중요한 요소이다. H.264는 압축 효율이 우수한 반면에 알고리즘이 지나치게 복잡하여 전력소비가 많은 것이 문제로 지적되어 왔다. 일반적으로 멀티미디어 디지털 시스템에서 잦은 메모리 접근이 전력소비 원인의 많은 부분을 차지한다. H.264 디코더도 메모리 접근 동작에서의 전력 소비가 H.264 디코더 시스템 전체 전력소비의 50% 이상을 차지한다는 보고가 있다[3]. 특히, 본 논문에서 제안하고자 하는 H.264 디코더를 위한 컨텍스트 적응형 가변길이 코딩(CAVLC) 모듈에서 4X4 블록을 복호화 할 때 계산을 위한 외부 테이블의 참조가 매우 빈번히 이루어지는 특징이 있다. CAVLC 디코더 설계에 있어서 메모리 접근을 대폭 줄임으로써 메모리 접근으로 인한 전력소비를 줄이기 위해 기존의 여러 가지 방법이 제안되어 왔다. Coeff_token 블록은 CAVLC 디코더를 구성하는 5 단계 중 한 단계에 해당한다. 본 논문에서는 메모리 참조 횟수를 획기적으로 줄여서 전력 소비를 최소화한 기존의 개선된 coeff_token 회로[4]를 선택하여 추가로 개선하였다. 본 논문에서는 [4]의 메모리구조와 수식 계산 방식을 단순화 시켜서 [4]와 전력 소비를 같은 수준으로 유지하면서도 면적을 줄이는 것을 목적으로 한다.

본 논문은 다음과 같은 구조를 가진다. 2 장에서는 기존의 저전력용 coeff_token 블록의 설계 방식을 소개하고 3 장에서는 이 회로를 저면적으로 개선한 새로운 설계를 제안한다. 4 장에서는 합성결과를 분석하고 결론을 제시한다.

2. 기존의 저전력용 CAVLC의 coeff_token 블록

본 장에서는 기존의 CAVLC 디코더 설계 연구 결과 중 전력 소비가 가장 낮은 [4]에서 구현한 coeff_token 블록의 구조를 재검토하고 회로의 면적을 줄이기 위한 개선사항을 찾아보고자 한다.

CAVLC 디코더는 인코딩된 입력 비트스트림(bitstream)으로부터 4x4 픽셀값을 나타내는 16개의 정수값을 복원해 내는 작업을 수행한다. CAVLC 디코더는 그림 1과 같이 coeff_token, sign of T1s, Level, Total_zeros, Run_before의 5 단계로 구성되어 있고 입력된 비트스트림으로부터 이 5단계를 거쳐서 4x4의 정

수 행렬을 완성해낸다[1,2]. 각 단계는 다음과 같은 수행 과정을 따르게 된다. 먼저 coeff_token단계에서는 4x4 블록에서 '0'이 아닌 계수의 개수(Tc)와 계수들 중 맨 뒤에 나타나는 '1'(Trailing 1)의 개수(T1s)를 찾아낸 다음, 이 '1'의 부호를 Sign of T1s 단계에서 결정한다. 이어서, Level 단계에서는 Tc-T1s 개수 만큼의 계수들의 값을 찾아낸다. 끝으로 Total_zeros단계에서 '0'아닌 계수들 사이에 삽입되는 '0'의 개수를 알아내고, Run_before단계에서 이 '0'들이 들어갈 위치 정보를 찾아내어서 최종적으로 4x4블록을 완성하게 된다.

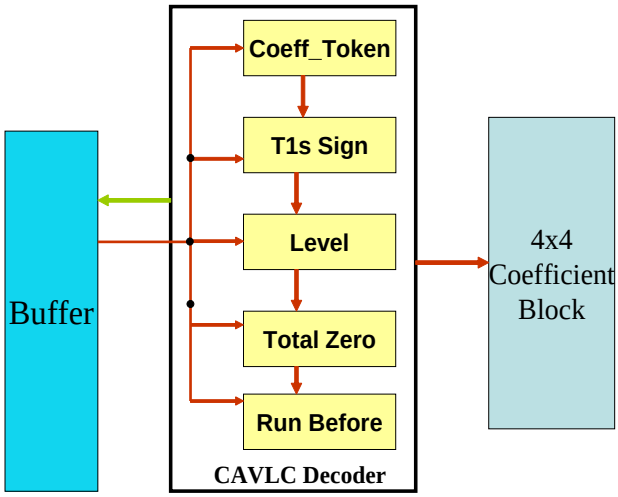

그림 1. CAVLC 전체 회로의 구조

표 1. H.264 표준의 VLCT0 표 (T1s과 Tc 값 정보)

T1s Tc	0	1	2	3
0	**1**	-	-	-
1	000101	**01**	-	-
2	00000111	000100	**001**	-
3	000000111	00000110	0000101	00011
4	0000000111	000000110	00000101	000011
5	00000000111	0000000110	000000101	0000100
. . .	. . .	. . .	. . .	. . .
15	0000000000000111	0000000000001010	0000000000001001	0000000000001100
16	0000000000000100	0000000000000110	0000000000000101	0000000000001000

본 논문은 CAVLC 계산을 위한 5개의 하위블록들 중 coeff_token 회로만을 대상으로 한다. coeff_token 단계에서는 인코딩된 비트스트림을 입력으로 받아서 Tc값과 T1s 값을 출력한다. Tc (Total Coefficient)는 4X4블록에서 '0'이 아닌 계수의 개수이고, T1s (Trailing '1')는 '0'이 아닌 계수의 맨뒤의 '1'의 계수의 개수이다(최대 3개까지). H.264 표준[1,3]에 의하면 다음 표 1의 VLCT 표를 이용하여 T1s과 Tc 값을 찾아낸다. 표준문서에는 이러한 VLCT표가 nC 값에 따라서 VLCT0, VLCT1, VLCT2, VLCT3의 네가지로 나뉘어 존재한다. 기존 연구 [4]에서는 표1에서 자주 참조하는 부분과 참조 빈도수가 낮은 부분을 나누어서 구현하였다. 가끔씩 발생하는 비트패턴에 대응하는 Tc1과 Ts 값을 찾기 위해서 그림 2와 같은 구조로 MBASE와 MCODE라는 두 개의 표를 이용하여 정보를 검색하도록 하였다. MCODE 표에서 읽혀진 값인 codenum은 최종적으로 식 (1)에 의해서 T1s과 Tc 값으로 변환된다.

T1s = codenum%4,
Tc = (codenum + 3 * T1s) >> 2. ..... 식 (1)

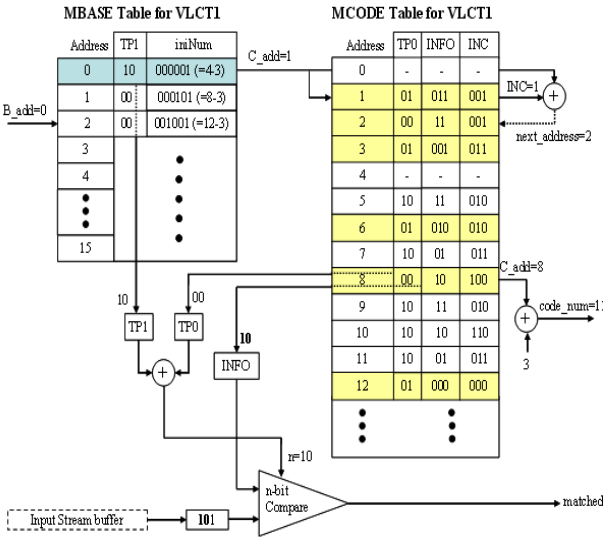

그림 2. Tc와 T1s 값을 위한 VLCT의 일부 구현

그림 2는, 입력 비트스트림에서 '1'앞에 나오는 '0'을 제외한 패턴이 "10"일때 codenum을 찾는 과정을 보여주고 있다. 그림 2에서 VLCT1을 구현하기 위해서 두 개의 표인 MBASE와 MCODE를 필요로 한다. MBASE는 매칭해야하는 입력코드의 비트수와 MCODE 표의 검색 시작 주소의 정보를 나타내는 표이고, MCODE는 T1s 값과 Tc 값을 계산하기 위한 codenum 값이 저장된 표이다.

그림 2는 [4]에서 coeff_token 단계에서 codenum와 MBASE 을 알아내기 위해서 사용한 VLCT1 표를 위

한 칩 내부의 메모리 구조를 보여준다. 기존[4]의 방법에서는 VLCT1 외에 VLCT0, VLCT2도 이와 같은 동일한 구조로 설계 되어 있어서 총 3 쌍의 테이블 참조용 메모리와 부가회로가 존재하게 된다. 결과적으로 회로의 구현시 칩내부 메모리를 위한 배선이 복잡해지고 연산기능이 표마다 중복되어 칩의 레이아웃 면적이 증가될 우려가 있다.

[4]에서는 전력소비가 많은 메모리 참조 횟수를 줄이기 위해서, 발생빈도가 높은 비트패턴에 대한 T1s와 Tc 값을 찾기 위해서는 VLCT 표를 그림 2처럼 테이블 참조로 찾는 방식으로 구현하는 대신에 산술 및 논리식을 이용한 "계산"에 의한 방식으로 구현한다. coeff_token 모듈의 저전력 설계에 필요한 이 계산을 위해서 [4]에서 사용된 주요 수식은 다음의 식 (2), 식 (3) 및 식 (4)이다. 이 식들에서 사용된 PCL은 비트스트림에서 '1'이 나타나기 전에 나타난 '0'의 개수를 의미한다.

```
T1s = PCL                              for VLCT0
    = 2  *  PCL + (1-In[0])            for VLCT1
    = 3 - In[2] * In[1:0]              otherwise
                                       ..... 식 (2)
Tc = T1s                               for VLCT0
   = T1s+PCL*(1-In[0])*(1-In[1])       for VLCT1
   = T1s+(1-In[2])<<2-(1-In[2])*In[1:0] otherwise
                                       ..... 식 (3)

B_addr = PCL + SEL_tab - 3             ..... 식 (4)
```

식 (2)와 식(3)은 표 참조 없이도 T1s와 Tc 값을 알 수 있도록 해주기 때문에 그림 2처럼 표 참조로 구현한 방식에 비해서 메모리 참조가 없기 때문에 전력 소비를 많이 줄여주는 주요한 부분이다. 식(4)는 MBASE의 시작주소 값을 계산하는 식이다.

그런데, 위 식(1) ~ 식(4)를 살펴보면 수식에서 곱셈 등 상대적으로 면적소비가 큰 연산자를 여러개 요구하고 있다. 이런 연산자의 개수를 줄일 수 있도록 기존의 수식 계산의 구현 방식을 개선한다면, 전력소비를 줄이면서도 면적소비를 줄이는 설계가 가능할 것이다.

3. 저면적을 위해 개선된 저전력 coeff_token 블록 설계

본 장에서는 2 장에서 소개한 기존 연구[4]의 저전력 coeff_token 회로에서 사용된 계산식과 메모리 구조를 개선하여 전력 소비양은 그대로 두면서 칩의 레이아웃 면적을 줄일 수 있는 구현을 하고자 한다.

식(5)는 식(1)에 사용된 연산자의 구현을 효과적으로 개선한 것이다. 식(5)는 식(1)과는 달리 Tc 값을 구하기 위한 식에서는 면적 소비가 큰 곱셈기를 덧셈기와 쉬프트 연산으로 대체함으로써 상대적으로 면적을 줄어들게 개선한 것이다. 그림 3은 codenum를 T1s와 Tc값으로 Converter하는 수식 (5)를 구현한 데이터 경로이다. 이 그림에서 matched 입력신호는 그림2의 비교기 출력값이다.

```
T1s = codenum[1:0]
Tc = {(codenum+T1s<<1)+T1s}>>2          식 (5)
```

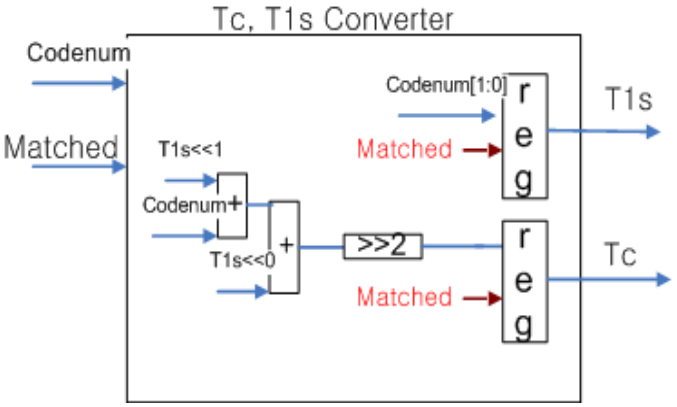

그림 3. 개선된 수식(5)에 의한 데이터 경로 블록도

아래 식 (6)과 식 (7)은, 2장의 식(2)와 식(3)을 개선한 것이다. 기존 수식에서의 뺄셈기의 역할을 2의 보수를 사용하여 덧셈기로 대체하여 다른 덧셈 연산과 회로를 공유하여 면적을 줄인다. 한편, 뺄셈을 요구하는 1-In[0], 1-In[1], 1-In[0]는 모두 1비트 연산인 점에 착안하여 ~In[0], ~In[1], ~In[2] 의 NOT 논리 연산으로 변경하여 산술 연산기를 쓰지 않고 간단히 구현하였다. 또한, 곱셈 연산도 모두 1비트와 2비트 간의 곱셈연산이므로 실제 구현에서는 곱셈기 대신 AND 게이트로 구현하였다.

```
T1s = PCL                      for VLCT0
    = (PCL<<1) + ~(In[0])      for VLCT1
    = 4 + ~(In[2]*In[1:0])     for otherwise
                               .....식 (6)
```

Tc = T1s for VLCT0
= T1s for ~In[0] and ~In[1] = 0
= T1s + PCL for ~In[0] and ~In[1] = 1 for VLCT1
= T1s + (~In[2]<<2)+~(~In[2]*In[1:0])+1 otherwise
..... 식 (7)

그림 4는 식(6)과 식(7)을 구현한 데이터 경로 설계 블록도를 나타낸 것이다. 이 그림에서 Sel_tab 신호는 VCLT0, VCTL1, VCTL2 중에서 선택할 때 지시해주는 신호이다.

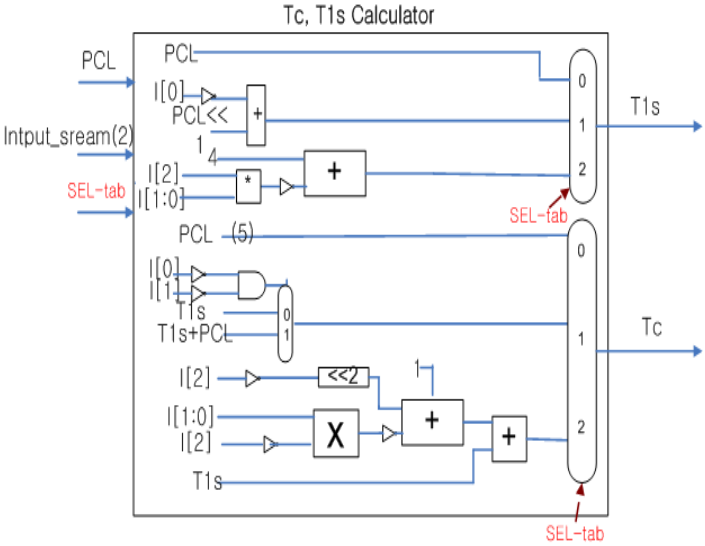

그림 4. 개선된 수식 (6), (7)에 의한 데이터 경로 블록도

그림 5 는 그림 1에서 소개한 VLCT0, VLCT1, VLCT2에 대한 테이블 참조 메모리(TLM) 구조를 나타낸 것이다. 기존에 테이블 구성은 MBASE와 MCODE가 각각 3 개씩 총 6개의 테이블로 구성 되어 있다. 이에 비해 본 논문의 방법에서는, 메모리 구성을 그림 6과 같이 하나의 메모리로 병합하여 모듈 내부의 라우팅 복잡도를 줄여서 전체 칩 면적을 줄이자는 제안이다.

그림 7는 개선된 MBASE와 MCODE를 위한 주소계산 모듈로서 식(4)를 그림 6의 구조에 맞는 변형한 구현이다. 이 그림에서 PCL은 입력 스트림에서 1이 나오기 전의 0의 개수이고, SEl_Tab은 nC값에 따라서 VLCT0, VLCT1, VLCT2 중에 선택하는 신호이다.

아래 그림 8은 본 논문에서 제안한 방법으로 설계된 coeff_token 전체 데이터 경로이다.

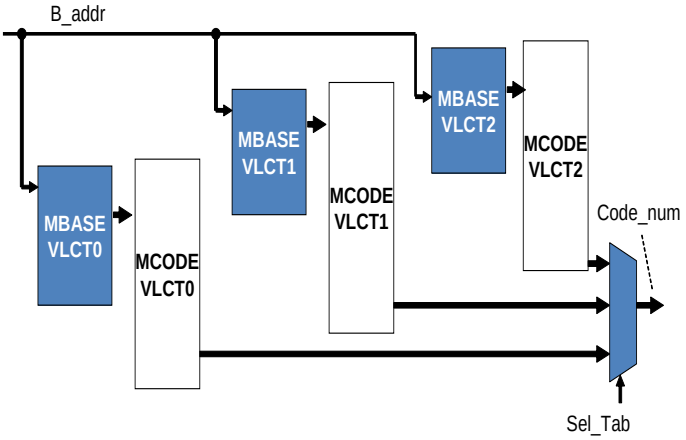

그림 5. [4]의 MBASE와 MCODE를위한 메모리구조

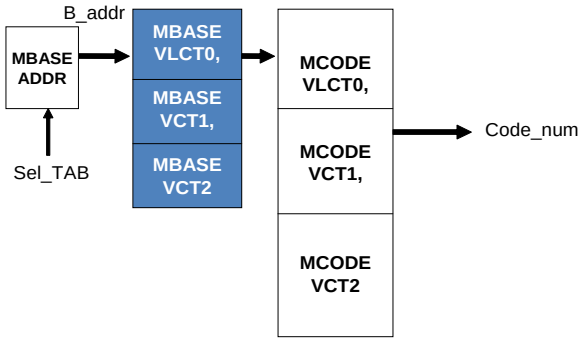

그림 6. 개선된 [4]의 MBASE와 MCODE를 위한 메모리 구조

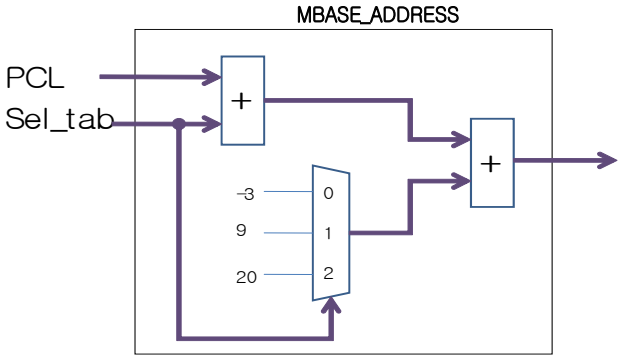

그림 7. Mbase 표를 참조하기 위한 base 주소 값계산 모듈

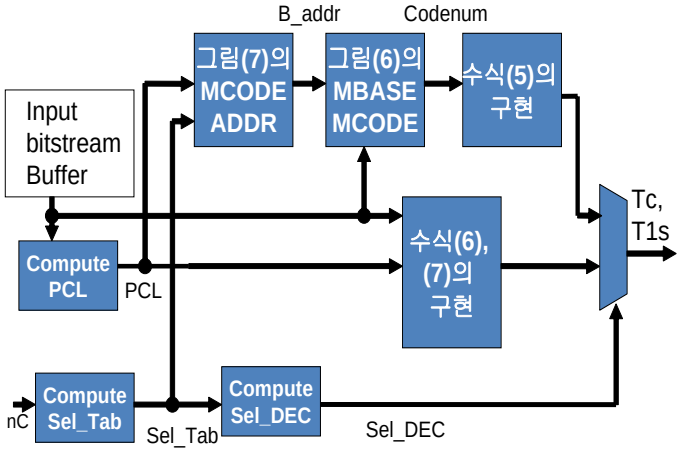

그림 8. 제안된 방법에 의한 coeff_token 모듈의 전체 데이터 경로 블록도 (그림 3,4,6,7의 종합)

4. 실험 결과 및 결론

본 논문에서는 설계 방식이 효과적인지 확인하기 위해서, 새로이 제안된 coef_tken 설계 방식과 기존[4]의 CAVLC coeff_token 회로 설계 방식을 각각 구현하여 면적을 비교하였다. 양쪽 회로는 모두 Verilog HDL을 사용하여 RT 수준에서 기술하였고, Modelsim을 이용하여 논리 시뮬레이션으로 기능을 검증하였다. 이 회로의 게이트 수준 구현은 삼성 0.18 um 공정을 대상으로 Synopsys 툴을 이용한 합성으로 수행되었다. 구현 결과 제안된 방식의 면적이 [4]의 기존 방식에 비하여 약 1.1% 줄어들었다. 그림 9은 이들의 면적 차이를 그래프로 보여주고 있다.

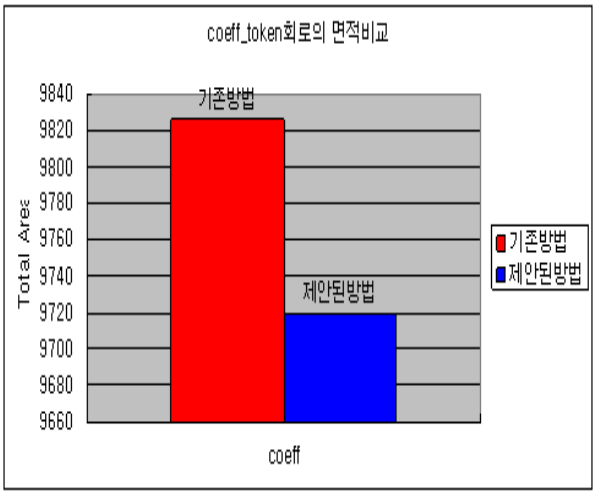

그림 9. 두가지 설계 방식에 따른 coeff_token 회로의 면적 비교 (삼성 0.18 um 공정)

향후에는 메모리 복잡도를 줄인 효과까지 정확히 측정하기 위해서 배치 및 배선과정을 모두 완료하여 레이아웃 단계까지 설계를 완성하고자 한다. 더불어 향후에는 CAVLC의 나머지 네연산 하위블록인, T1s sign, total zero, level, run_before를 모두 설계를 완성하여 CAVLC 디코더 전체에서의 본 접근법의 효과도 확인하고자 한다.

참고 문헌

[1] H.264 and MPEG-4 Video Compression Video Coding for Next-generation Multimedia Iain E. G. Richardson

[2] Yong Ho Moon; Gyu Yeong Kim; Jae Ho Kim; "An efficient decoding of CAVLC in H.264/AVC video coding standard" Vol. 51, ,Aug. 2005, pp. 933-938

[3] Draft ITU-T Recommentdation and Final Draft International Standard of Joint Video Specification(ITU-TRec.H.264|ISO/IEC14496-10AVC)

[4] Yong Ho Moon "A New Coeff-Token Decoding Method With Efficient Memory Acess in H.264/AVC video coding standard" Vol. 17, June 2007, pp. 729-736