논문 2008-45SD-7-9

H.264/AVC의 효율적인 파이프라인 구조를 적용한 CABAC 하드웨어 설계

(Efficient Pipeline Architecture of CABAC in H.264/AVC)

최 진 하*, 오 명 석**, 김 재 석*

(Jinha Choi, Myungseok Oh, and Jaeseok Kim)

요 약

본 논문에서는 최신 동영상 압축 기술인 H.264/AVC (Advanced Video Coding)에서 엔트로피 코딩 방법 중 하나로 사용되는 CABAC (Context Adaptive Binary Arithmetic Coding)의 하드웨어 구현과 부호화 처리율을 높이기 위한 알고리즘 및 구조를 제안한다. CABAC는 CAVLC에 비해 최대 15%까지 더 나은 압축효율을 낼 수 있는 장점을 가지고 있지만 연산의 복잡도는 훨씬 높아 진다. 특히 부호화 과정 중 데이터 사이의 의존도가 높기 때문에 연산과정의 복잡도가 더욱 증가하게 된다. 따라서 연산양을 줄이기 위한 다양한 구조가 제안되었으나, 여전히 데이터의 의존도에 의한 부호화에 latency가 존재하게 된다. 본 논문에서는 이진 산술 부호화의 첫 단계인 확률 값을 계산하는데 필요한 range의 7, 8번째 비트를 빠르게 계산하는 구조와 부호화할 심벌이 MPS인 경우 부호화 단계를 한 단계 줄일 수 있는 구조를 제안하였다. 제안된 구조를 적용하여, 6가지 시퀀스에 대하여 실험한 결과 기존의 구조에 비해 약 27-29%의 수행시간을 줄일 수 있었다. 또한 제안된 구조를 하드웨어로 구현한 결과 0.18um standard library에서 19K gate를 사용하였다.

Abstract

In this paper, we propose an efficient hardware architecture and algorithm to increase an encoding process rate and implement a hardware for CABAC (Context Adaptive Binary Arithmetic Coding) which is used with one of the entropy coding ways for the latest video compression technique, H.264/AVC (Advanced Video Coding). CABAC typically provides a better high compression performance maximum 15% compared with CAVLC. However, the complexity of operation of CABAC is significantly higher than the CAVLC. Because of complicated data dependency during the encoding process, the complexity of operation is higher. Therefore, various architectures were proposed to reduce an amount of operation. However, they have still latency on account of complicated data dependency. The proposed architecture has two techniques to implement efficient pipeline architecture. The one is quick calculation of 7, 8th bits used to calculate a probability is the first step in Binary arithmetic coding. The other is one step reduced pipeline architecture when the type of the encoded symbols is MPS. By adopting these two techniques, the required processing time was reduced about 27-29% compared with previous architectures. It is designed in a hardware description language and total logic gate count is 19K using 0.18um standard cell library.

Keywords : H.264/AVC, Entropy coding, CABAC, Arithmetic Coding, VLC coding

* 정회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)
** 정회원, LG전자
(LG Electronics)
※ 본 연구는 지식경제부가 지원하는 국가 반도체 연구개발 사업인 "시스템 집적 반도체 기반 기술개발 사업 (시스템IC2010)" 및 교육과학기술부 BK21 사업의 일환인 연세대학교 전기전자공학부 TMS 사업단의 지원을 통해 개발된 결과임을 밝힙니다.
접수일자: 2008년3월12일, 수정완료일: 2008년7월7일

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 표준으로 기존의 표준과 비교해 뛰어난 압축 성능으로 인해 현재 가장 각광 받는 압축 표준으로 압축률을 높이기 위해 새로운 기법들을 제시하였다[1]. 특히 엔트로피 부호화

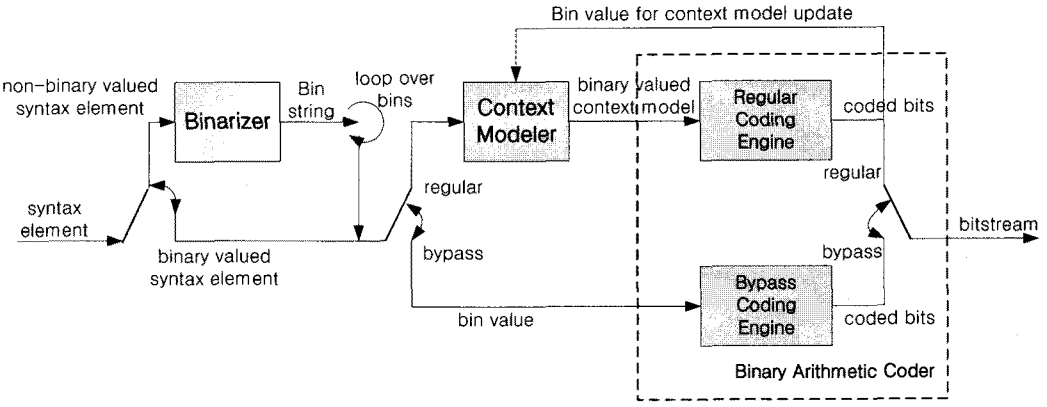

그림 1. CABAC부호화기의 하드웨어 블록 다이어그램
Fig. 1. Hard-Ware block diagram of CABAC coder.

과정에서는 기존 압축표준과 다른 방법을 제시하였다. H.264/AVC는 압축의 최종단계인 엔트로피 부호화 과정에서 CAVLC (Context-Adaptive Variable Length Coding)와 CABAC (Context-Adaptive Binary Arithmetic Coding)를 사용한다. CAVLC는 DCT 계수만을 부호화하는데 사용되며, 계수와 연속되는 '0'의 길이 등, 복호에 필요한 정보를 부호화 한다. 이것은 기본적으로 테이블에 기반으로 하여 효율적으로 부호화하는 방법이다. CABAC는 기본적으로 산술부호화방식을 사용하는데 이는 기본적인 산술부호화 방식에서 점점 발전되어 보다 효율적인 방법을 사용하는 방식이다. CABAC는 CAVLC에 비해 최대 15%까지 더 나은 압축효율을 낼 수 있지만 연산의 복잡도는 훨씬 높아지게 된다[2]. CABAC는 구문요소를 이진화한 후, 각 비트(bit)의 확률값에 대해서 연산을 수행하기 때문에 많은 연산과정이 요구되며, 확률값이 클 경우 재정규화 과정에서 여러 번의 반복수행이 일어나게 된다. 이때, 이진 산술 부호화에 필요한 데이터와 재정규화 과정에 필요한 데이터 사이의 의존관계를 가지고 있기 때문에 비효율적인 연산과정을 수행한다. 부호화의 한 사이클에 두 개 이상의 부호화를 수행하기 어렵고, 연산 과정이 순차적인 구조를 가져야만 한다. 따라서 고속의 CABAC 부호화를 수행하는데 어려움을 가진다[3]. 본 논문에서는 CABAC의 알고리즘을 분석하여, 연산의 복잡도를 가져오는 부분을 해소하고, 보다 빠르고, 자원의 소모를 줄일 수 있는 여러 최적화 방법들에 대해 제안할 것이다.

본 논문의 구성은 다음과 같다. 먼저 Ⅱ장에서는 CABAC의 부호화 과정 및 하드웨어 설계시 문제점에 대하여 살펴보고 Ⅲ장에서는 이진 산술 부호화의 효율적인 파이프라인 단계를 위해 제안된 알고리즘과 하드웨어 구조를 제시한다. 그리고 Ⅳ장에서는 기존에 제시된 구조와 비교분석을 하고, 마지막 Ⅴ장에서는 본 논문의 결론을 맺는다.

Ⅱ. CABAC 개요 및 하드웨어 설계시 문제점

2.1. CABAC 부호화 과정

CABAC 부호화는 슬라이스 단위의 출력으로 이루어지고 하나의 슬라이스의 인코딩이 끝나면 마지막에 종결 데이터를 추가하여 전송한다. 하나의 슬라이스는 많은 구성요소들로 이루어졌으며 CABAC의 입력으로 들어간다. CABAC는 CAVLC와는 달리 구문요소(syntax element)라 불리는 매크로블록과 관련된 정보도 부호화를 진행한다. CABAC는 입력 받은 구문요소가 이진화(binarization) 과정이 필요한지 아닌지를 검사하며, 이진화 과정이 필요하면 이진 데이터로 변환하여 이진 산술 부호화를 한다. 이때, 구문요소가 '0' 과 '1' 의 빈도가 동일한 데이터라면 확률값을 이용하지 않는 간결한 우회 이진 산술 부호화(Bypass 모드)로 부호화한다[4].

다음 그림 1은 CABAC부호화기의 하드웨어 블록 다이어그램을 보여준다. CABAC는 구문요소를 이진화하는 Binarizer 블록, 확률값을 생성하는 Context Modeler 블록, 실제 압축이 일어나는 Regular Coding Engine 블록, 그리고 동일한 확률값을 가질 때 수행되는 Bypass Coding Engine 블록으로 나뉜다. 구문요소를 입력으로 받은 CABAC 부호화기는 먼저 이진수 여부를 판단하고, 이진수가 아닌 구문요소에 대해서 Binarizer 블록에서 이진화 과정을 수행한다. 이진화가 끝난 데이터들은 이진화된 각 심벌이 동일한 확률값을 가지는지 아닌지를 확인한 후, 동일한 확률값을 가지지 않는 경우 문맥

색인 (ctxIdx)과 이진수 값(binVal)을 입력으로 받아 Regular Coding Engine 블록에서 이진 산술 부호화를 시작한다. 만일 부호화할 심벌이 동일한 확률값을 가진다면, Bypass Coding Engine 블록에서 확률값의 참조 없이 우회 이진 산술 부호화를 수행한다[5].

2.2 하드웨어 설계시 문제점

CABAC 부호화에서 고속의 부호화를 방해하는 가장 큰 부분은 Regular Coding Engine 블록에서 수행하는 이진 산술 부호화 과정이다. 이진 산술 부호화 과정에서는 컨텍스트 모델링의 롬 테이블로부터 확률값을 읽어와 확률구간(range) 와 경계구간(low)를 업데이트 과정과 재정규화 루프과정으로 이루어져 있다. 먼저, 이진 산술 부호화 과정의 경우에는 확률값 rLPS, 확률구간 range, 경계구간 low의 값을 이용하여 연산이 수행되는데 이 값들은 상호 의존적인 관계를 가지고 있다. 따라서 하드웨어로 설계 시에 이런 의존성으로 인해 많은 제약을 가지게 된다. 그림 2와 같이 low와 range를 업데이트 하기 위해서는 두 가지의 연산이 수행되는데 MPS인 경우에는 2개, LPS인 경우에는 3개의 연산이 수행된다. 이때, 각 레지스터의 의존성을 줄이기 위해서 빠른 연산을 통해 재정규화부로 업데이트된 range와 low값을 넘겨줘야 한다. 재정규화부에서는 일정수준이하로 range가 떨어졌을 때, 이 값을 다시 일정수준 이상으로 올려 줄때까지 쉬프트 연산을 반복 수행된다[6].

따라서 루프의 끝나는 시점을 예측하기 어렵고 루프의 반복으로 인해 상당한 지연이 발생된다. 또한 재정규화 내부에 bitsOutstanding과 관련된 반복 루프를 가지고 있고 이는 최종 출력비트와 관련이 있다. 최종 출력은 low와 bitsOutstanding의 값에 의존해서 발생되는데 bitsOutstanding의 값을 예측하기 어렵기 때문에 최종 출력은 상당한 지연을 가지게 된다. 이런 재정규화

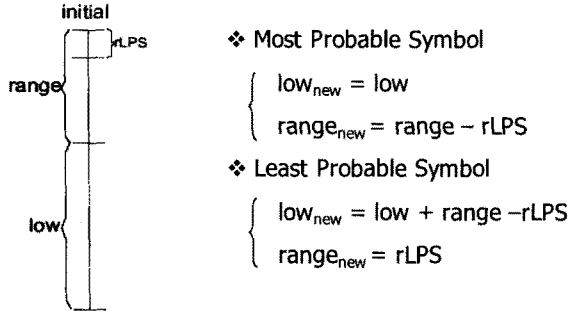

그림 2. 발생확률에 따른 range와 low의 업데이트 과정
Fig. 2. Range and Low update process each Probable.

의 루프와 재정규화 내부의 비트 생성 루프는 업데이트된range와 low값의 빠른 전달을 저해한다. 이는 곧 부호화의 파이프라인의 수행을 어렵게 만들고, 부호화 속도에 현저한 영향을 미치게 된다. 따라서 부호화 속도를 개선하기 위해서는 데이터간의 의존성을 줄이고, 기존의 재정규화 과정과 출력 비트 생성부에서 발생하는 루프를 효율적으로 처리하는 것이 중요하다. 두 과정의 루프를 효율적으로 처리하지 못하면, 현재 입력 심벌의 부호화가 끝나는 시점까지 다음 심벌의 부호화가 진행될 수 없고 CABAC 부호화를 고속으로 처리할 수 없기 때문이다.[7~8]

Ⅲ. 제안된 CABAC 구조

3.1. 부호화 단계의 효율적인 파이프라인 구조

확률상태(pStateIdx)와 range[7:6]을 이용하여 확률값(rLPS)을 계산한 후, 부호할 심벌에 따라 확률값을 이용하여 range를 업데이트한다. 업데이트된 range의 재정규화를 위해 range의 첫 번째 '1"의 위치를 탐색하여 재정규화 반복횟수를 계산하고, 계산된 재정규화 반복횟수 만큼 업데이트된 range를 왼쪽으로 쉬프트연산을 함으로써 range를 일정 수준이상으로 유지되게 만든다. 쉬프트 되어진 range의 7번째와 8번째 비트는 다음 심벌의 확률값을 메모리로부터 참조하기 하기 위해 사용된다. 이는 재정규화를 거치기 이전의 range는 일정수준 이하의 값을 가질 수 있는 불완전한 값이기 때문에 LOD를 이용하여 왼쪽 쉬프트를 함으로써 재정규화 과정을 끝낸 range값을 비로써 메모리를 참조하는데 이용할 수 있다.

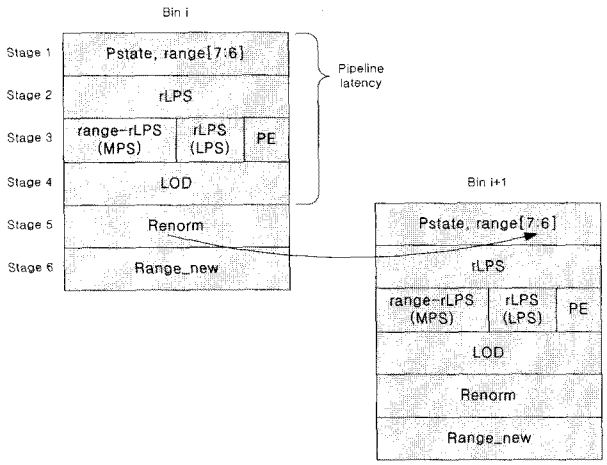

그림 3. 세부적인 단계의 파이프라인 구조
Fig. 3. More detailed pipeline architecture.

이와 같은 의존관계는 그림 3의 화살표에 나타난다. 이와 같은 의존관계로 인해 다음 심벌 부호화의 시작은 이전 심벌의 부호화의 5단계에서 시작되어지고, 파이프라인 latency는 4단계를 가지게 된다. 본 절에서는 위 그림 3과 같이 실제적인 연산과정에서 발생하는 파이프라인 latency를 줄이기 위해 range의 7, 8번째 비트를 빠르게 예측할 수 있는 구조를 제안한다.

다음 그림 4는 range의 7, 8번째 비트를 예측하기 위해 제안된 구조를 보여준다. 부호화의 첫 단계인 IS단계의 연산 과정을 끝낸 후, 업데이트된 range는 LOD블록으로 보내진다. LOD블록은 range의 첫 번째 '1'의 위치를 탐색하여 그 값을 이용하여 IS단계에서 업데이트된 range를 왼쪽 쉬프트 시킨다. 쉬프트 되어진 range의 7번째와 8번째 비트는 다음 심벌의 확률값을 메모리로부터 참조하기 위해 사용된다. 그러나 그림 4에서 왼쪽 쉬프트 되기 이전의 IS단계에서 업데이트된 range와 쉬프트 된 range 사이에서 상관관계가 있음을 알 수 있다. 쉬프트 되기 이전의 range에서 첫 번째 '1'의 위치에서부터 마지막 비트까지의 값은 변하지 않고 쉬프트 되어진 range에 그대로 남아있음을 알 수 있다. 이와 같은 상관관계를 이용하여 재정규화 과정을 통해 구해진 range의 7번째와 8번째 비트를 예측할 수 있다. 위의 그림 4에서와 같이 IS단계 이후의 range를 2비트씩 8개의 구간으로 나눈다. 이는 재정규화 과정에서 반복 횟수는 0-7이기 때문이다. LOD가 0일 경우에는 쉬프트 되기 이전 range의 7, 8번째 비트가 확률값 (rLPS)을 메모리로부터 참조하기 위해 전달되어지고, LOD의 값에 따라 range내 적절한 위치의 2비트를 메모리 참조를 위해 이용한다. LOD가 7인 경우에는 range의 마지막 비트와 '0'을 적용한다. 이는 재정규화를 위해 왼쪽 쉬프트를 하였을 경우 range의 마지막 비트 이후에는 '0'으로 값이 채워지기 때문이다. 위 그림 4에서 보면 IS단계를 수행한 range의 값은 '0 0001 1101' 이다. LOD블록에서 첫 번째 '1'의 위치를 탐색한 후, 재정규화 반복횟수를 나타내는 LOD는 4의 값이 된다. 이때, range의 값이 4만큼 쉬프트 되고, 동시에 재정규화 되어진 range의 7번째와 8번째 비트를 예측한다. LOD가 4일 경우에는 재정규화 되기 전의 3번째와 4번째를 예측된 값으로 이용한다. 따라서 재정규화 되어진 range의 7, 8번째 비트가 '11'이 됨을 예측할 수 있다. 실제 왼쪽 쉬프트 되어진 range의 7, 8 번째 비트와 비교해보면 두 값이 동일함을 알 수 있다. 이와 같은 구조를 적용하여 range의 7, 8 번째 비트를 기존 구조보다 한 단계 빠르게 예측함으로써 파이프라인의 latency를 줄일 수 있다. 다음 그림 5는 현재 부호화하는 심벌과 다음 부호화할 심벌이 부호화될 때, 제안된 구조의 파이프라인 과정을 보여준다.

IS단계에서의 연산과정을 거친 후, 메모리를 참조하기 위해 사용되는 range의 7, 8번째 비트를 예측하기 위해 2비트씩 8가지 경우의 수로 나눈다. range의 첫 번째 '1'의 위치를 탐색한 LOD는 값을 출력하고, 이 값은 8가지 경우의 수 중 하나를 선택한다. LOD에 의해 선택되어진 값은 IS 단계에서 갱신된 range를 왼쪽 쉬프트하는데 사용된다. 따라서 메모리로부터 확률값을 참조하기 위해 요구되는 range의 7,8 번째 비트는 기존의 구조에서는 재정규화 되어진 range값에 의존하였던 반면에 제안된 구조에서는 재정규화 과정에서 왼쪽 쉬프트가 일어나기 전 range의 첫 번째 '1'의 위치를 구하는 과정인 LOD에 의존하게 된다. 이런 관계는 그림 5의 화살표로 나타나있다. 이와 같이 데이터 사이의 의존관계를 줄임으로써 제안된 구조는 기조의 구조에 비

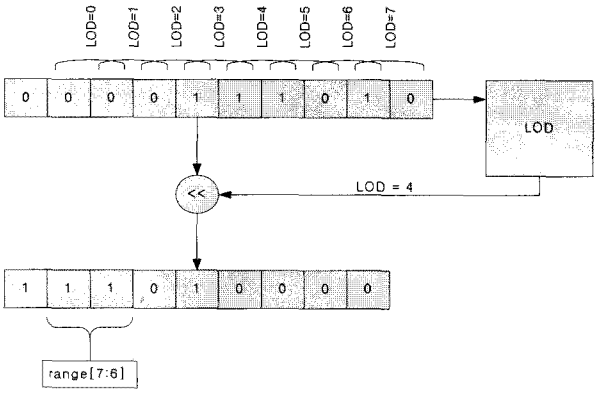

그림 4. Range의 7, 8번째 비트를 예측하기 위한 구조
Fig. 4. 7, 8th bit of range prediction architecture.

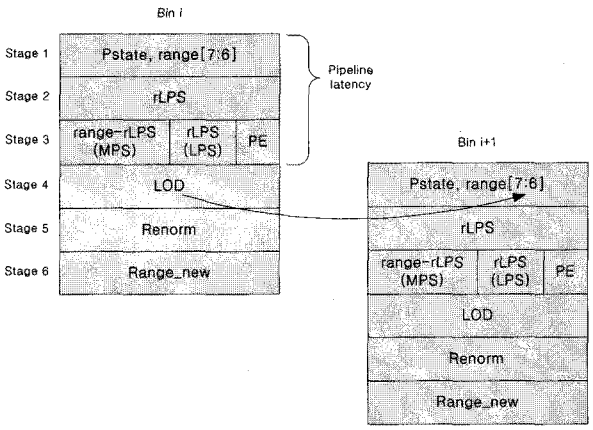

그림 5. 7, 8번째 비트 예측을 이용한 파이프라인구조
Fig. 5. Pipeline architecture using bit prediction.

해 다음 심벌의 부호화를 한 단계 빨리 시작할 수 있는 파이프라인 구조를 이루었고, 파이프라인 latency는 4단계에서 3단계로 1단계가 줄어들었다.

3.2. LPS 부호화를 위해 제안된 룩업 테이블

앞서 설명한 기존의 6단계보다 한 단계 적은 5단계로 이루어져 있음을 알 수 다. 이는 IS단계에서 부호화할 심벌이 LPS인 경우에는 추가적인 덧셈이나 뺄셈의 연산과정을 요구하지 않기 때문이다. 심벌이 LPS인 경우, range를 업데이트하기 위해서 덧셈과 뺄셈의 연산과정을 수행하는 대신 확률값(rLPS)을 range값으로 대체함으로써 range를 업데이트하게 된다. 따라서 이진 산술 부호화의 단계가 기존의 6단계에서 5단계로 줄어든다. 또한 MPS의 경우 파이프라인 latency의 한 단계였던 덧셈, 뺄셈의 단계가 생략됨으로써 파이프라인 latency 역시 3단계에서 2단계로 줄어든다. 그러나 이런 구조 또한 2단계 파이프라인 latency를 가지는 비효율적인 파이프라인 구조를 가진다[9]. 따라서 본 절에서는 완전한 파이프라인 구조를 만들기 위해 다음과 같은 방법을 제안한다.

확률값(rLPS)은 확률상태(pStateIdx)와 재정규화된 range의 7, 8번째 비트를 이용하여 메모리로부터 참조하여 구해진다. 이때, 확률상태는 0부터 63까지 64가지의 경우를 가지며, 확률 구간을 나타내는 range의 7, 8번째 비트는 4가지의 경우의 수를 가진다. 이는 확률구간을 4개의 구간으로 다시 나눈 후, 각 구간에 따라 64가지의 확률값을 다르게 적용하기 때문이다. 따라서 rLPS를 참조하기 위한 메모리는 64x4 개의 값을 저장하고 있다. 확률상태와 range의 7, 8번째 비트를 이용한 256가지 경우에 대한 확률값이 존재한다. 이러한 확률값을 분석해 보면 전체의 확률구간을 4개의 구간으로 나눈 구간 중 아래쪽에 위치한 구간(range의 7, 8번째 비트가 0)에서의 확률값이 동일한 확률상태에서 다른 구간의 확률값에 비해 적고, 가장 위쪽에 위치한 구간(range의 7,8번째 비트가 3)에서의 확률값이 동일한 확률상태에서 다른 구간의 확률값보다 큰 것을 알 수 있다. 확률값의 최대값은 확률상태가 0이고, range의 7, 8번째 비트가 3일 때, 0x0f0(240)의 값을 가지게 되고, 최소값은 확률상태가 63일 때, 4가지 구간에 대해서 모두 0x002(2)의 값을 가진다. 이와 같은 확률 표를 이용하여 확률값(rLPS)을 참조하는 것과 동시에 참조된 확률값의 첫 번째 '1'의 위치를 예측할 수 있다.

표 1에서 각 열은 확률구간의 4가지 구간을 나타내고

표 1. 확률 상태에 따른 LOD 값
Table 1. LOD of each probability.

	range[7:6]			
LOD	0	1	2	3
1	0-2	0-6	0-9	0-12
2	3-15	7-19	10-22	13-25
3	16-29	20-32	23-36	26-38
4	30-42	33-46	37-49	29-52
5	43-56	47-60	50-62	53-62
6	57-62	61-62	x	x
7	63	63	63	63

각 행은 각 확률값에 대한 첫 번째 '1'의 위치를 나타낸다. 표의 안의 값은 나누어진 확률구간에 대해서 LOD의 값에 대응되는 확률상태의 값들을 나타내고, LOD의 7가지 값들은 확률값(rLPS)에 따라 다음과 같이 나눠진다. rLPS의 값이 0x080(128)보다 클 때의 LOD는 '1', 0x040(64) 보다 크고 0x080(128) 보다 작을 때의 LOD는 '2', 0x020(32) 보다 크고 0x040보다 작을 때의 LOD는 '3', 0x010(16) 보다 크고 0x020(32) 보다 작을 때의 LOD는 '4', 0x008(8) 보다 크고 0x010(16) 보다 작을 때의 LOD는 '5', 0x004(4) 보다 크고 0x008(8) 보다 작을 때의 LOD는 '6', 0x002(2) 보다 크고 0x004(4) 보다 작을 때의 LOD는 '7'이 된다. 이와 같이 예측 되어진 LOD는 부호화할 심벌이 LPS인 경우 기존의 3단계에서 계산된 LOD와 동일한 값을 갖는다. 이는 IS단계에서의 range를 계산하기 위한 추가적인 연산과정이 없고, rLPS값을 range 값으로 대체하기 때문에 재정규화 전의 range 값에서 첫 번째 '1'의 위치를 탐색하는 LOD와 같은 값을 갖게 된다. 따라서 위와 같은 rLPS의 LOD를 예측하는 구조를 적용함으로써 우리는 부호화할 심벌이 LPS인 경우 부호화 단계를 줄이고 완전한 파이프라인 구조를 적용할 수 있다.

그림 6은 부호화할 심벌이 LPS인 경우 제안된 구조가 완벽한 파이프라인 구조로 동작함을 보여준다. 부호화할 심벌이 LPS인 경우에 부호화의 첫 단계는 확률상태(pStateIdx)와 range의 7, 8번째 비트를 이용하여 메모리로부터 확률값(rLPS)을 참조하고, 동시에 제안된 구조를 이용하여 rLPS에 대한 LOD를 계산한다. rLPS와 LOD 값이 동시에 계산되기 때문에 LOD를 계산하기 위한 단계가 필요하지 않고, 재정규화를 위해 이전 단계에서 계산된 rLPS와 LOD를 이용하여 왼쪽 쉬프트를 수행한다. 이로써 한 심벌에 부호화 단계는 기존의 5단계에서 4 단계로 다시 줄어든다. 또한 다음 심벌을 부호화하기 위해 필요한 재정규화된 range의 7, 8번째

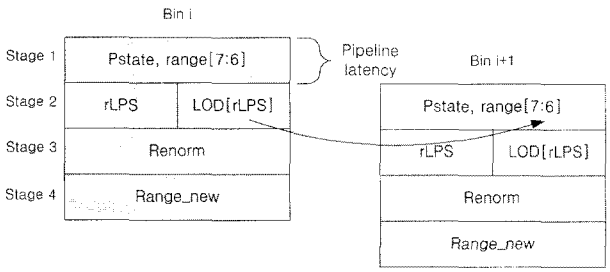

그림 6. rLPS의 LOD 예측을 통한 파이프라인 구조
Fig. 6. Pipeline architecture with LOD prediction of rLPS.

비트는 이전 절에서 제안된 구조를 적용하면 LOD값에 의존하는데 LOD값이 rLPS와 동시에 수행되기 때문에 파이프라인 latency가 2단계에서 1단계로 줄어든다. 따라서 심벌의 부호화가 완전한 파이프라인 구조를 이루게 된다.

Ⅳ. 기존 구조와의 비교 분석 및 하드웨어 구현

제안된 알고리즘의 성능을 분석하고 Li[10]와 Osorio[11]의 구조와 비교를 수행하기 위해서 Container, Foreman, News, Mobile, Highway, Akiyo 등의 6가지 시퀀스를 이용하였으며, 각각의 시퀀스에 대해 QP 값 12, 20, 28로 설정하여 실험을 수행하였다. 그 외의 나머지 조건들은 동일한 값을 가지도록 하였다.

표 2는 MPS, LPS, EQ 심벌의 연산과정에 따라 소요되는 사이클 수와 전체 심벌 당 사이클 수를 제안된 구조와 기존의 구조에 대해 비교한 것이다. 먼저, MPS인 경우에는 재정규화 과정이 반복되는 구간에 대해서는 약 24%의 감소가 있었고, 재정규화가 일어나지 않는 구간에 대해서는 동일한 사이클 수를 가졌다. 다음 LPS인 경우에는 항상 재정규화 과정이 일어나기 때문에 재정규화 과정이 발생하는 경우에만 약 50% 감소가 있었다. 마지막으로 EQ 심벌의 경우, EQ 심벌과 EQ가 아닌 심벌이 부호화되는 경우에는 EQ가 아닌 심벌과 동시에 부호화를 하기 때문에 사이클 수가 요구되지 않고, EQ 심벌의 쌍에 대해서는 동일한 사이클 수가 요구된다. 따라서 사이클 수의 감소는 없었다. 이와 같은 심벌의 특성에 따라 요구되는 사이클 수를 이용하여 심벌 당 요구되는 사이클 수를 비교하여 보면, 기존의 구조가 2.5222를 필요로 하는데 비해 제안된 구조는 1.8222를 필요로 하였고, 약 27-29%의 감소가 있었다.

표 3은 6가지 seqeunce를 QP와 Prediction type에 따라 Frame당 평균 Cycle 수를 보여준다. Li[10]와

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

Type of the encoded symbol		Previous[11][10] cycles/symbol	Proposed cycles/symbol	Ratio
MPS	renorm	1.8/1.8	1.35	24%
	Non-renorm	0.2/0.2	0.2	0
LPS	renorm	0.5/0.5	0.25	50%
	Non-renorm	0/0	0	0
EQ	EQ+non-EQ	0.0778/0	0	0
	EQ's pairs	0.0222/0.0222	0.0222	0
Total		2.6/2.5222	1.8222	27% ~29%

표 3. 시퀀스와 예측 타입에 따른 평균 사이클 수
Table 3. Average Cycle with Each prediction type.

QP 12	Osorio[11]		Li[10]		Proposed	
	Intra	Inter	Intra	Inter	Intra	Inter
Container	282867	78492	284459	78811	229055	57784
Foreman	287272	143521	288875	147713	234742	107580
News	274692	67104	276299	67411	222428	49006
Mobile	634281	250162	637809	251567	504529	182137
Highway	230478	194433	231826	195458	189875	142750
Akiyo	187860	24582	188899	24653	153660	18126
QP 20	Osorio[11]		Li[10]		Proposed	
	Intra	Inter	Intra	Inter	Intra	Inter
Container	154800	18330	155528	18387	116208	13528
Foreman	153903	50773	154659	50989	114560	37127
News	155735	22979	156491	23079	116713	16757
Mobile	365134	124434	366885	125068	274755	90985
Highway	112106	67206	112570	67450	84782	49841
Akiyo	104621	3746	105111	3755	77900	2748
QP 28	Osorio[11]		Li[10]		Proposed	
	Intra	Inter	Intra	Inter	Intra	Inter
Container	73048	5209	73387	5218	53253	3869
Foreman	73112	20398	73430	20464	53127	15022
News	79717	9424	80079	9459	57947	6921
Mobile	191669	51245	193214	51438	140131	37638
Highway	37753	13725	37886	13768	27446	10172
Akiyo	53426	1411	53648	1413	31340	1047

Osorio[11]의 구조와 제안된 구조의 사이클 수를 비교해 보면, Li[10]와 Osorio[11]의 구조에 비해 약 20%-27%정

도의 사이클 수가 줄어듦을 알 수 있다. 이는 MPS, LPS, EQ의 비율이 일정하지 않고 Inter prediction에서는 QP값이 작아질수록 LPS의 비율이 증가하고, Intra prediction에서는 QP값이 작아질수록 MPS와 EQ의 비율이 증가하기 때문이다. 따라서 Intra prediction의 경우 QP값이 클수록 제안된 구조의 효율이 커지고, Inter prediction의 경우 QP값이 작을수록 제안된 구조의 효율이 커진다. 위의 비교 영상에서 QP값 12을 가지는 mobile 영상의 경우 가장 큰 사이클 수를 필요로 한다. 이 경우 실시간으로 영상을 부호화하기 위해서 제안된 구조는 최대 15MHz의 클럭 주파수를 필요로 하는데 비해 Li[10]와 Osorio[11]의 구조는 20MHz의 클럭 주파수를 필요로 한다. 이는 동일한 영상을 각각의 클럭 주파수에서 부호화할 때, 제안된 구조는 Li[10]와 Osorio[11]의 구조에 비해 단위 시간 당 전력소모를 약 20%-21% 줄이는 효과가 있다.

표 4는 하드웨어 구현 결과를 다른 구조들과 비교한 것으로, 제안된 구조의 하드웨어 복잡도는 Osorio[11]의 구조와 거의 동일하고, binarization과 context modeling 부분을 예측한 Li[10] 구조의 전체적인 복잡도보다는 약 7% 증가함을 알 수 있다. 각 블록별 하드웨어 복잡도를 비교해보면, 제안된 구조는 Binarization 블록에서는 gate counts가 줄어들었지만 나머지 블록에서는 gate counts가 모두 증가 하였다. Binarization 블록에서는 제안된 구조는 룩업테이블을 참조하여 이진화를 수행하는 경우에 요구되는 룩업테이블의 대칭성을 이용하여 테이블의 사이즈를 줄임으로써 Osorio[11]의 구조 보다 약 750 gate counts가 줄어들었다. 그러나 Context Modeling의 경우는 Osorio[11]의 구조가 소프트웨어를 이용하여 문맥색인을 초기화하고 업데이트하는데 비해 제안된 구조에서는 하드웨어로 구현하였다. 따라서 문맥색인을 초기화하기위해서 하드웨어 복잡도가 증가하였다. 이진 산술부호화의 블록에서는 rLPS의 첫 번째 '1'의 값을 계산하기 위해 제안된 구조와 range의 7, 8번째를 예측하기 위해 제안된 구조로 인해 하드웨어 복잡도가 증가하였다. Bit Generator 블록에서는 low를 이용하여 왼쪽 쉬프트 되어진 값을 버퍼에 저장하고 버퍼에 저장된 값을 이용하기 위해 여러 연산과정을 거치기 때문에 하드웨어 복잡도가 증가하였다.

표 4. 하드웨어 구현 결과 비교
Table 4. Comparison of Hardware Implemetation Results.

Unit	Osorio[11]	Li[10]	Proposed
Binarization	6500	6800*	5750
Context modeling	5420	5900*	5735
Range	2238	4140	2378
Low	2438		2498
Bit generator	2830	830	2990
Total	19426	17670*	19054

* Estimated results

V. 결 론

본 논문에서는 CABAC의 고속처리를 위한 구조를 제안하였다. CABAC는 CAVLC에 비해 최대 15%까지 더 나은 압축효율을 낼 수 있는 장점을 가지고 있지만 연산의 복잡도는 훨씬 높다. 부호화 과정 중 데이터 사이의 의존도가 높기 때문에 연산과정의 복잡도는 더욱 증가하게 된다. 따라서 연산양을 줄이기 위한 다양한 파이프라인 구조가 제안되었으나, 여전히 데이터의 의존도에 의한 부호화에 latency가 존재하게 되고, 파이프라인의 latency를 증가 시키게 된다. 따라서 이와 같은 파이프라인 latency를 줄이기 위해 확률값을 결정하는데 중요한 데이터인 range의 7, 8번째 비트를 Li[10]와 Osorio[11]의 구조보다 한 단계 빠르게 예측할 수 있는 구조를 제안하였다. 또한, 부호화할 심벌이 LPS인 경우 range는 확률값으로 업데이트 되어진다. 이때, range의 7, 8번째 비트와 문맥색인의 확률상태를 이용하여 재정규화의 반복 횟수를 빠르게 예측함으로써 부호화 단계를 한 단계 줄이고, 완전한 파이프라인 구조를 이루도록 설계하였다. 제안된 구조를 적용하여 실험을 수행한 결과, Li[10]와 Osorio[11]의 구조에 비해 CABAC 부호화의 사이클 수를 약 27%-29%를 줄일 수 있었고 Li[10]와 Osorio[11]의 구조보다 단위시간 당 약 20%-21%의 전력소모를 줄일 수 있었다. 제안된 구조는 0.18um 표준 라이브러리를 기준으로 합성한 결과 약 19K gate의 logic gate를 사용하였다.

참 고 문 헌

[1] T. Wiegand, G. J. Sulivan, G. Bjontegaard and A. Luthra, "Overview of the H.264/AVC Video Coding Standard", IEEE Trans. on Circuits and Systems for Video Technology, Vol. 13, No. 7, pp 560-576, Jul. 2003.

[2] D. Marpe, H. Schwarz and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard",

IEEE Trans. on Circuits and Systems for Video Technology, Vol. 13, No. 7, pp 620-636, Jul. 2003.
[3] H. Shojaina and S. Sudharsanan, "A VLSI Architecture for High Performance CABAC Encoding", Visual Communication and Image Processing in Proceedings of the SPIE, pp. 1444-1454, Jul. 2005.
[4] D. Marpe, H. Schwarz, G. Blattermann, G. Heising and T. Wiegand, "CONTEXT-BASED ADAPTIV BINARY ARITHMETIC CODING IN JVT/H.26L", International Conference on Image Processing, pp. 513-516, Sep. 2002.
[5] XH. Tian, M. Le. Thinh, B. L. Ho, "A CABAC Encoder Design of H.264/AVC with RDO Support", 18th IEEE/IFIP International Workshop on Rapid System Prototyping, pp 167-173, May. 2007.
[6] R. R. Osorio and J. D. Bruguera, "A New Architecture for fast Arithmetic Coding in H.264 Advanced Video Coder", 8th Euromicro Conference on Digital System Design, pp 298-305, Sep. 2005.
[7] K. Andra, T. Acharya and C. Chakrabarti, "A MUTI-BIT BINARY ARITHMETIC CODING TECHNIQUE", International Conference on Image Processing, pp. 928-913, Sept. 2000.
[8] 윤재복, 박태근, "H.264/AVC를 위한 파이프라인 이진 산술 부호화기 설계", 대한전자공학회지 논문지 제 44권 SD편, 제 6호, pp 514-521, 2007년 6월.
[9] R. R. Osorio and J. D. Bruguera, "Arithmetic Coding Architecture for H.264/AVC CABAC Compression System", Euromicro Symposium on Digital System Design, pp. 62-69, Sep. 2004.
[10] L. Li, Y. Song, T. Ikenmaga and S. Goto, "A CABAC Encoding Core with Dynamic Pipeline for H.264/AVC Main Profile", IEEE Asia Pacific Conference on Circuit and Systems, pp. 760-763, Dec. 2006.
[11] R. R. Osorio and J. D. Bruguera, " High Throughput Architecture for H.264/AVC CABAC Compression System", IEEE Trans. on Circuits and Systems for Video Technology, Vol. 16, No. 11, pp 1376-1384, Nov. 2006.

저 자 소 개

최 진 하(정회원)
2005년 연세대학교 전기전자 공학부 학사 졸업.
2007년 연세대학교 전기전자 공학과 석사 졸업.
2007년~현재 연세대학교 전기전자공학과 박사 과정
<주관심분야 : H.264/AVC, ISP, SoC 설계>

오 명 석(정회원)
2006년 경희대학교 전자공학과 학사 졸업.
2008년 연세대학교 전기자공학과 석사 졸업.
2008년~현재 LG전자 MC 사업부 연구원
<주관심분야: H.264/AVC, ISP, SoC 설계>

김 재 석(정회원)
1977년 연세대학교 전자공학과 학사 졸업.
1979년 한국과학원 전기 및 전자공학과 석사 졸업.
1988년 Rensselaer Polytechnic Institute 전자공학과 박사 졸업.
1993년~1995년 한국 전자통신 연구원 책임 연구원.
1995년~현재 연세대학교 전기전자공학과 교수.
<주관심분야: 통신 및 영상 시스템, VLSI 신호처리, 임베디드 S/W 및 SoC 구현>