JANT 한국항행학회논문지
Journal of Advanced Navigation Technology

기반 기술

J. Adv. Navig. Technol. 18(6): 600-608, Dec. 2014

H/W-S/W 병행설계를 이용한 CABAC의 효율적인 하드웨어 구현

An Efficient Hardware Implementation of CABAC Using H/W-S/W Co-design

조 영 주 · 고 형 화*
광운대학교 전자통신공학과

Young-Ju Cho · Hyung-Hwa Ko*
Department of Electronics and Communication Engineering, KwangWoon University, Seoul 139-701, Korea

[요 약]

본 논문에서는 CABAC (context adaptive binary arithmetic coding)를 하드웨어로 구현하기 위하여 병행설계 (co-design) 기법을 사용하였다. H.264/AVC의 부호기 전체를 C언어로 개발하고, CABAC만을 하드웨어 IP로 설계하고, H.264/AVC의 나머지 부분은 소프트웨어로 설계하였다. CABAC의 문맥모델러 부분을 하드웨어로 설계하여 연산 값을 지속적으로 업데이트시킴으로써 메모리를 효율적으로 사용하고 스트림을 절감시키는 설계를 하였다. 설계된 IP는 Xilinx ML410보드의 Virtex-4 FX60 FPGA에 다운로드하여 MicroBlaze CPU를 이용하여 H.264/AVC의 참조 소프트웨어인 JM과 연동하도록 설계하였다. 기능 시뮬레이션은 ModelSim을 이용하였다. 기존의 CABAC 하드웨어 모듈이 레지스터 레벨에서 설계하여 개발기간이 오래 걸리는 데 비하여 본 논문의 설계 기법은 소프트웨어 엔지니어가 쉽게 하드웨어를 개발하는 것이 가능해지는 장점이 있으며 설계시간도 짧다. 또한, 동일한 방법으로 구현된 CAVLC모듈과 Slice 사용량을 비교해볼 때, 1/3 이하로 감축됨을 보였다. 본 연구에서 제시한 개발 방법은 임베디드 환경에서 고성능 동영상 압축 부호화시 하드웨어 가속기가 필요한 부분을 설계할 때 유용할 것으로 보인다.

[Abstract]

In this paper, CABAC H/W module is developed using co-design method. After entire H.264/AVC encoder was developed with C using reference SW(JM), CABAC H/W IP is developed as a block in H.264/AVC encoder. Context modeller of CABAC is included on the hardware to update the changed value during binary encoding, which enables the efficient usage of memory and the efficient design of I/O stream. Hardware IP is co-operated with the reference software JM of H.264/AVC, and executed on Virtex-4 FX60 FPGA on ML410 board. Functional simulation is done using Modelsim. Compared with existing H/W module of CABAC with register-level design, the development time is reduced greatly and software engineer can design H/W module more easily. As a result, the used amount of slice in CABAC is less than 1/3 of that of CAVLC module. The proposed co-design method is useful to provide hardware accelerator in need of speed-up of high efficient video encoder in embedded system.

Key word : High efficiency video coding, Context adaptive binary arithmetic coding, H.264, M-coder.

http://dx.doi.org/10.12673/jant.2014.18.6.600

Received 28 November 2014; **Revised** 1 December 2014
Accepted (Publication) 24 December 2014 (30 December 2014)

***Corresponding Author; Hyung-Hwa Ko**

Tel: +82-2-940-5137
E-mail: hhkoh@kw.ac.kr

Ⅰ. 서 론

영상 네트워크 산업의 발전으로 영상의 공급과 수요가 폭발적으로 증가하고, full HD급 이상의 고화질 영상에 대한 수요 역시 꾸준히 증가하고 있다. 따라서 H.264/AVC 영상 압축 표준보다 효율이 더 뛰어난 영상 압축 표준이 요구된다. 이에 따라 ITU-T의 비디오 코딩 전문화그룹 (video coding experts group, VCEG)과 ISO/IEC의 동화상 전문가 그룹 (moving picture experts group, MPEG)이 공동으로 설립한 JCT-VC (join collaboration team on video coding)에서 H.264/AVC표준 보다 두 배 이상의 압축률을 목표로 하는 HEVC (high efficiency video coding)를 2013년 1월 차세대 최종 표준안으로 승인 하였다[1],[2].

그림 1에 HEVC 부호화기의 블록도를 보였다. H.264/AVC에서 사용 가능한 엔트로피 부호화 방법으로는 CABAC (context-based adaptive binary arithmetic coding)와 CAVLC (context-based variable length coding) 두 가지 모두 가능하지만, HEVC에서는 압축 효율이 더 높은 CABAC 만을 채택하여 사용하기로 정하였다[3],[4]. 그림2에는 H.264/AVC의 CABAC 부호기 블록 다이어그램을 보였다.

HEVC에 유일한 엔트로피 부호화 방식으로 CABAC가 채택되면서 산술 부호기의 구현의 중요성이 높아지고 있다. H.264/AVC 및 HEVC에 사용되는 CABAC의 이진산술부호화 방법은 M-coder라 불리는 산술부호기이다[5].

산술부호기를 하드웨어로 구현한 연구는 CABAC 전체를 구현하거나[6]. 혹은 BAC (binary arithmetic coding) 구현에 관한 연구가 있다[7],[8]. CABAC는 이전 데이터를 재사용하여 처리하여야 하는 알고리즘의 계산적 복잡성으로 인해 다른 부호화 방법에 비해 부호화기의 복잡성이 높은 반면 탁월한 압축 성능을 보여주고 있기 때문에 많은 양의 데이터로 이루어진 최근의 멀티미디어 컨텐츠를 처리하는데 필수적이라 하겠다. 기존의 산술부호기의 설계방법으로는 레지스터 레벨에서 VHDL언어를 이용하여 최소 사이즈의 모듈을 개발하는데 초점이 맞추어져 있다.

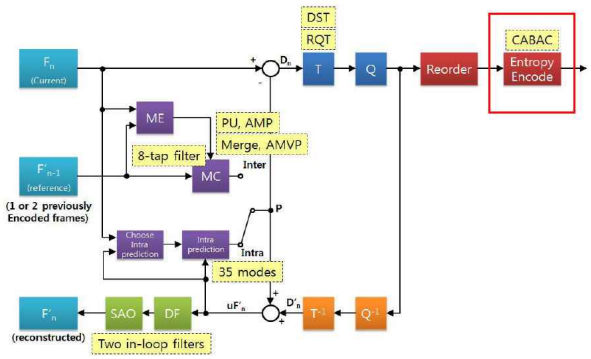

그림 1. HEVC 부호화기 블록 다이어그램
Fig. 1. Encoder block diagram of HEVC.

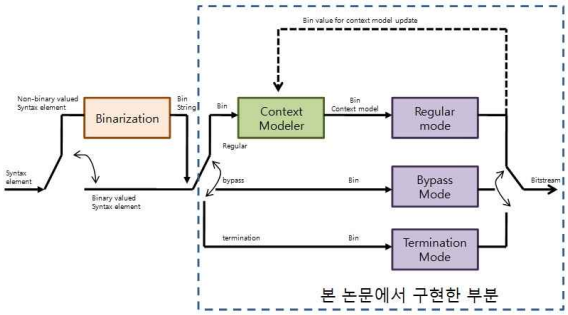

그림 2. H.264/AVC 부호기의 CABAC 블록 다이어그램
Fig. 2. CABAC encoder block diagram of H.264/AVC.

그러나 레지스터 레벨에서 하드웨어 개발을 수행함으로써 훈련받은 전문 인력에 의해 개발되어야 하고, 개발에 시간이 많이 드는 단점이 있다.

이에 본 논문에서는 H.264/AVC 전체 부호기를 C 언어로 개발하고, CABAC 부호기를 하드웨어 모듈로 개발하는 방법을 제안하고자 한다. 본 논문은 고속 비디오코딩부의 여러 모듈가운데 하드웨어 가속기가 필요한 모듈(예; ME)을 하드웨어로 설계하는 병행설계 방법을 사용하면 개발시간을 단축시키고, 개발된 회로의 수정이 용이하다. 특히, 이 방법은 하드웨어 개발에 익숙하지 않은 소프트웨어 엔지니어가 쉽게 개발할 수 있는 장점이 있다.

본 논문의 구성은 다음과 같다. 2장에서는 이진 산술 부호화에 대한 개요와 설계한 IP에 대한 이론적 설명을 하였으며, 3장에서는 하드웨어 설계 방법과 알고리즘에 대해 기술하였다. 4장에서는 설계한 IP의 설계 검증과 성능을 분석하였고, 5장에서 결론을 이끌어 내었다.

Ⅱ. 산술 부호기

2-1 산술부호기의 개요

산술부호화는 무손실 압축에 사용되는 엔트로피 부호화 알고리즘 가운데 하나로써 허프만 부호화가 각각의 기호를 1:1로 부호를 대체하는 반면에 산술부호화는 전체 메시지를 하나의 실수 n $(0.0 \leq n < 1.0)$으로 대체한다. 산술 부호화는 주어진 기호와 확률 분포에 대해 최적의 압축을 만들 수 있다[9].

산술부호기의 원리를 간단히 알아보면, 산술부호기는 각 심벌들의 발생확률에 비례하여 구간을 만들어 주는데, [0,1]의 구간의 일부로 표현한다. 예를 들어, 그림3에서 3개의 심벌이 있을 경우, 각각 발생확률이 0.8, 0.02, 0.18이라고 가정 할 때, 만약 입력데이터가 "1321"의 순서로 발생하고 최초의 심벌이 "1"이라면, 발생확률에 비례하여 [0,0.8]구간을 설정한다. 이어서, "3"이 들어오면 [0,0.8]구간을 확률에 비례하여 [0.656, 0.8]로 구간을 재정의 한다.

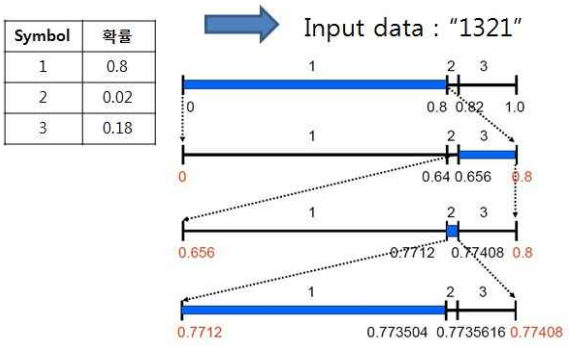

그림 3. 기본적인 산술부호기의 부호화 단계
Fig. 3. Coding phase of basic arithmetic coder.

이와 같은 방법으로 입력이 연속으로 들어오면 구간을 더 작은 구간으로 나눈다. 이와 같은 과정을 반복하여 구간을 세분화하고, 최종 심벌이 들어올 때, 구간에 속하는 가장 짧은 이진데이터가 최종 출력이 된다.

산술 부호화의 장점은 심벌들 간의 발생확률 값이 크게 차이가 날 때 압축효율이 좋다. 예를 들어, 이진문서의 경우 흑색과 백색의 발생빈도가 크게 차이가 나므로 효율적인 압축이 가능하다. 반면에 허프만 부호화는 이러한 이진 문서에 대하여 압축률이 높지 못하다. 그러나 산술부호화는 압축 대상 심벌의 개수가 많을수록 압축데이터량이 엔트로피에 근접하게 되지만, 각 심벌을 압축할 때 마다 구간을 계산해야 하는 복잡함 때문에 연산량이 많다는 단점이 있다. 또한 그림 3에서처럼 심벌을 처리할 때마다 정밀도가 증가하기 때문에 사용 메모리 또한 늘어나게 된다. 또한 메시지의 끝을 나타내는 별도의 기호를 사용하거나 몇 개의 기호가 들어있는지에 대한 부가적인 정보가 필요하다는 문제가 있기 때문에, 단순한 시퀀스의 경우에는 오히려 허프만 부호화 보다 효율이 떨어질 수 있다.

이러한 단점을 줄이기 위해 제안된 산술부호기로 IBM에서 개발한 Q-coder[9]가 있는데 재정규화(renormalization)을 적용하였고, 최초로 곱셈을 없앤 산술부호기이다. Q-coder는 구간(Range)의 값이 1일 경우에 이를 이용하여 곱셈이 없는 산술코더를 가능하게 했다. 또한 압축된 데이터의 생성 완료된 바이트 안으로의 캐리 전파(carry propagation)를 제한하기 위해 필요한 비트 스터핑(bit stuffing) 과정을 포함하고 있다. 또 하나 주목할 것은 Q-coder에서는 재정규화가 수행될 때마다 확률 추정을 수행하는데, 간단한 확률 테이블을 정의하여 구현하였다. Q-coder는 오늘날 산술부호화의 발전과 연구에도 큰 영향을 주었다. Q-coder 이후 코딩 효율을 향상시킨 QM-coder와 MQ-coder가 차례로 개발되었으며, M-coder는 이러한 QM-coder와 MQ-coder와 달리 곱셈 기능을 룩업테이블로 대체하여 하드웨어 구조를 간단하게 하면서 성능을 개선시켰다[10]. M-coder는 기존의 산술부호기의 단점을 보완하기 위해 고안된 E1, E2, E3 scaling을 기반으로 구현되었으며, 각 scaling별로 다음과 같이 동작된다.

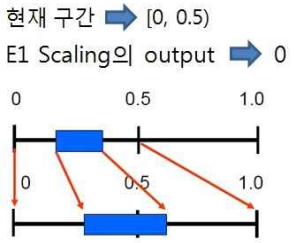

그림 4. E1 스케일링
Fig. 4. E1 scaling.

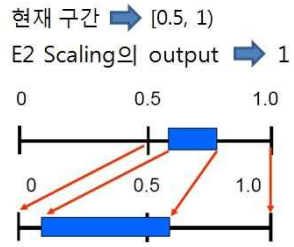

그림 5. E2 스케일링
Fig. 5. E2 scaling.

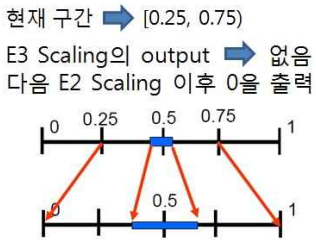

그림 6. E3 스케일링
Fig. 6. E3 scaling.

즉, 현재의 구간이 실수 0.5를 기준으로 0.5이하의 확률에 속하는 경우에는 그림 4와 같은 E1 scaling, 0.5이상의 확률에 속하는 경우에는 그림 5와 같은 E2 scaling, 0.5가 현재 구간에 함께 포함되는 경우는 그림 6과 같은 E3 scaling을 이용한다. 구간의 코딩 크기와 정밀도를 유지하기 위해서 간격을 분할 한 후 재 정규화(renormalization)를 해 준다.

2-2 이진 산술 M-coder

M-coder(modulo coder)는 H.264/AVC 비디오 코딩 표준에 배포된 최초의 성공적인 이진산술 부호화 방식이며, MQ-coder보다 좋은 압축률과 적은 계산량을 갖고 있다. 본 논문에서 구현한 H.264/AVC의 CABAC 부호화기는 이진화, 문맥 모델러, 이진 산술 부호화기의 3단계로 구성으로 되어있다[11],[12]. 기존의 연구들은 주로 이진산술부호기의 설계로 초점을 맞추었으나 본 논문에서는 이진산술부호기 M-coder와 문맥 모델러 부분을 하드웨어로 설계함으로써 FPGA를 보다 효율적으로 사용할 수 있도록 설계하고자 한다.

표 1. State인덱스와 codIrange인덱스에 따른 rangeLPS 테이블

Table 1. RangeLPS table of stateIdx and codIrangeIdx.

State-Idx	CodIrangeIdx				State-Idx	CodIrangeIdx			
	0	1	2	3		0	1	2	3
0	128	176	208	240	32	27	33	39	45
1	128	167	197	227	33	26	31	37	43
2	128	158	187	216	34	24	30	35	41
3	123	150	178	205	35	23	28	33	39
4	116	142	169	195	36	22	27	32	37
5	111	135	160	185	37	21	26	30	35
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
27	35	43	51	59	59	7	8	10	11
28	33	41	49	56	60	6	8	9	11
29	32	39	46	53	61	6	7	9	10
30	30	37	43	50	62	6	7	8	9
31	29	35	41	49	63	2	2	2	2

이진화된 신호는 이진 값의 확률에 따라 정규화 모드, 바이패스 모드, 종결 모드의 세 가지 모드로 부호화를 수행한다. 이진 값을 갖는 신택스 요소가 입력 신호로 주어질 경우에는 이진화가 생략되고 바이패스 모드로 동작된다.

정규화 모드에서는 부호화할 신택스 요소들을 문맥모델링을 거쳐 확률 모델이 결정 된 후 결정된 확률 모델과 주어진 확률 모델에 따라 빈(bin) 값은 실제 산술 부호화 모듈에 입력 값으로 주어진다. 이후 산술 부호화를 수행하고 지속적으로 확률 모델을 업데이트 한다. 그림 7과 같이 정규화 모드는 주어진 빈 값에 해당하는 문맥 모델의 stateIdx와 valMPS(최대 빈도 심볼) 값을 사용하여 수행한다. codIrange와 offset을 계산하기 위해 256 < codIrange < 512에 해당 하는 범위를 4단계로 균일하게 나눈 표 1과 같이 해당 양자화 값을 가진 rangeLPS 테이블에 접근하여 codIrange를 계산한다. offset은 빈 값과 valMPS값을 비교하여 동일한 경우 기존 값을 유지하며 동일하지 않은 경우 codIrange를 더한 값을 가진다. codIrange와 offset은 다음 확률 값을 적용하기 위해 transIdxLPS와 transIdxMPS의 ROM을 이용하여 업데이트를 수행하는데 빈 값과 valMPS값을 비교하여 동일한 경우는 transIdxMPS를 사용하며 동일하지 않은 경우에는 transIdxLPS를 이용하여 stateIdx와 valMPS값을 업데이트 한다.

바이패스 모드는 주어진 빈 값에 대해 바이패스 부호화 모드를 선택할 때 사용되며 이전에 부호화된 신택스 요소의 확률 모델을 사용하지 않는 부호화 방식이다. 종결 모드는 매 슬라이드의 종결 여부를 판단하는 end_of_slice_flag의 이진 값을 이용하여 처리한다. 바이패스 모드는 정규화 모드에서 사용하는 확률 평가와 업데이트 과정을 수행하지 않고 구간 재분할을 수행할 때 두 심볼의 발생 확률이 동일(0.5의 확률)하다고 가정하여 구간을 재분할하므로 부호화 속도가 빠르다. offset의 계산은 우선 왼쪽 시프트 하여 해당 값을 2배 증가한 후 입력된 빈 값에 따라 다른 계산을 한다. 빈 값이 '1'인 경우 offset는 codIrange값을 연산하여 '0'인 경우는 값이 변경되지 않는다[13].

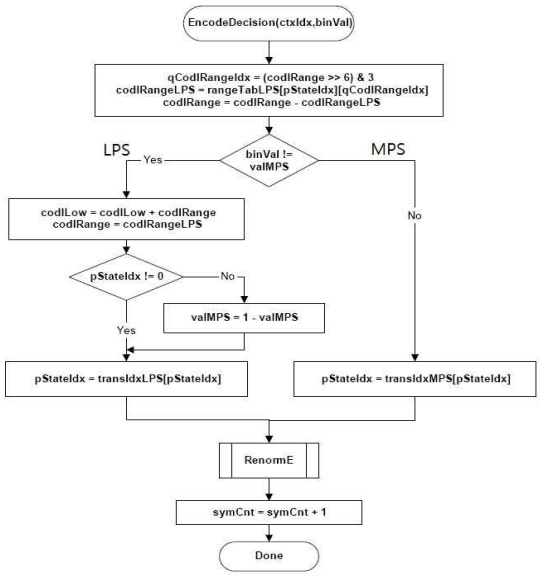

그림 7. 정규화 모드

Fig. 7. Normalization mode.

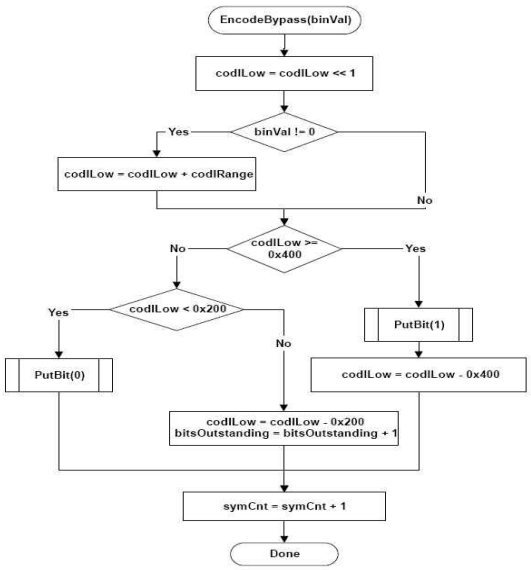

그림 8. 바이패스 모드

Fig. 8. Bypass mode.

그림 8은 바이패스 모드의 수행 과정을 나타낸다.

종결모드는 매 매크로블록마다 있는 end_of_slice_flag의 이진 값을 부호화 한다. 그림 9에서 보듯 종결모드는 바이패스 모드와 마찬가지로 확률평가와 업데이트 과정을 수행하지 않아 비교적 연산이 간단하다. 이진 산술 연산을 연속적으로 수행하면, 현재의 확률 구간을 표현하기 위해 많은 비트가 필요하게 된다. 이러한 문제를 해결하기 위해 codIrange가 일정 수준 이하로 떨어지게 되면 재정규화를 수행하게 된다.

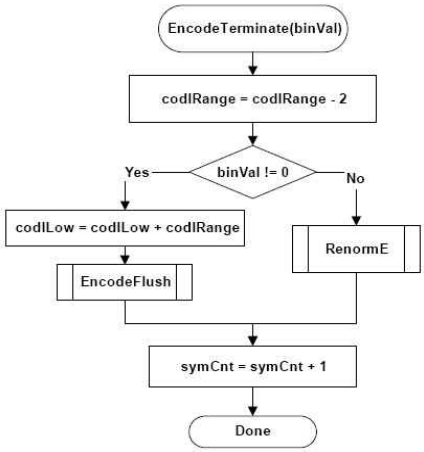

그림 9. 종결 모드
Fig. 9. Terminate mode.

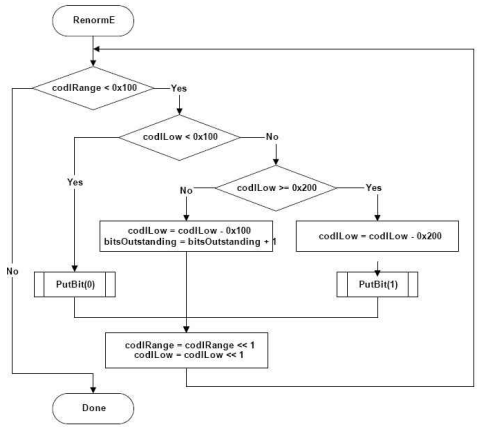

그림 10. 재정규화
Fig. 10. Renormalization.

그림 10과 같이 재정규화는 codIrange가 0x100이상이 되면 재정규화 과정을 수행하지 않으며 codIrange가 0x100보다 작은 경우에만 재정규화 과정을 수행한다. 이 과정에서 codIrange 값은 왼쪽 시프트를 하여 2배가 되며, codIrange는 최대 7번 반복 수행 한다[13].

Ⅲ. 산술부호기 하드웨어 설계

3-1 병행 설계 기법

하드웨어와 소프트웨어를 동시에 설계하는 병행설계(co-design)는 설계하고자 하는 시스템의 연산량이나 제어의 편리성 등에 따라 소프트웨어 혹은 하드웨어로 블록을 나누어 설계하는 방식이다[14],[15]. 이때 가장 먼저 고려해야 할 부분은 소프트웨어와 하드웨어의 분할 기준이다. 분할 기준은 큰 계산량을 필요로 하지 않으면서 복잡한 연산을 수행해야 하는 경우에는 소프트웨어로 구현하고, 큰 계산량, 규칙적인 연산, 반복적으로 사용되는 연산, 병렬적으로 처리될 수 있는 연산은 하드웨어로 구현하는 것이 유리하다. 다음으로 고려해야 할 부분은 하드웨어와 소프트웨어를 연결하는 인터페이스이다. 인터페이스 부분은 하드웨어와 소프트웨어간의 데이터 전달을 담당하게 되며, 효율적인 인터페이스 블록을 설계해야 하드웨어와 소프트웨어간의 오버헤드(overhead)를 줄일 수 있다. 하드웨어 블록과 소프트웨어 블록간의 인터페이스 방법은 메모리를 공유하는 방법, FIFO를 사용하는 방법, 핸드쉐이킹 프로토콜을 이용하는 방법 등이 있다[15].

본 논문에서는 병행 설계를 쉽고 효과적으로 하기 위하여 Impulse사에세 만든 CoDeveloper 프로그램을 이용하였다[14]. CoDeveloper 프로그램은 전체부호기를 C/C++언어로 개발하고 하드웨어부와 소프트웨어부로 분할한 뒤, 하드웨어 부는 C 소스를 VHDL/Verilog 언어로 변환해 주는 툴이다. Xilinx FPGA 내부에 있는 PowerPC 혹은 MicroBlaze를 이용한 소프트웨어 연산과 하드웨어의 연동을 용이하게 해주고 설계초기 단계에서부터 병행설계를 고려한 설계를 할 수 있도록 지원해주고, 스트림 설계가 유연하기 때문에 S/W와 H/W간의 스트림을 설계한 후, 향후 스트림의 수정 시 수정이 용이하다는 장점이 있다. 이러한 장점을 활용하여 FPGA에 최적화된 스트림을 설계 가능하며 본 논문에서도 이러한 장점을 이용하여 최적의 스트림을 설계하였다.

이 툴을 이용하여 H.264/AVC 부호기에서 CABAC 부호화는 하드웨어로 제작하였고 기타 다른 모듈은 소프트웨어로 처리할 수 있도록 설계하였다. 이 방법을 이용할 경우 가격대 성능비를 향상 시킬 수 있으며 하드웨어와 소프트웨어의 설계 시간, 설계비용 등을 감소시킬 수 있다. 또한, 기존의 하드웨어 설계 방법이 VHDL 또는 Verilog에 익숙한 전문인력에 개발을 의존하고 있는 반면에 병행설계 기법은 C/C++에 익숙한 소프트웨어 엔지니어들도 C를 통해 하드웨어 모듈을 쉽게 개발할 수 있는 장점이 있다.

본 논문에서는 2가지 방법으로 설계하였다. 첫째로, BAC(M-coder) IP를 하드웨어로 설계하고, BAC에서 처리할 신택스와 각각의 컨텍스트를 H.264/AVC 모듈에서 스트림으로 전송하는 방식으로 설계하였다. 두 번째로, H.264/AVC의 CABAC 부분을 하드웨어로 설계하였다. BAC에 필요한 컨텍스트는 하드웨어에 저장하는 방식으로 설계하였고, 소프트웨어 모듈에서는 신택스와 동작모드 및 출력스트림을 스트림으로 전송하는 방식으로 설계하였다. 이 두 방법 모두 본 논문에서 제안한 방법으로 두 방법의 실험결과를 제시하였다.

설계한 H/W IP는 Modelsim을 통하여 시뮬레이션 검증을 하였고, Xilinx ISE/XPS를 사용하여 H/W를 IP로 추가 한 후, synthesis와 implementation을 수행하였다. XPS를 이용하여 bit 파일이 생성되면 Xilinx사의 개발용 보드인 ML410보드의 Virtex-4 FX60 FPGA에 다운로드 하였다.

3-2 CABAC 하드웨어 설계

1) BAC H/W 설계 (방법1)

그림 11은 병행설계를 통해 구현된 H.264/AVC의 BAC 모듈인 M-coder를 연동시킨 H.264/AVC를 대략적인 구조를 나타낸다. 그림에서와 같이 H.264/AVC 부호기에서 BAC모듈만 하드웨어로 설계하고 나머지 모듈들은 S/W로 구현한 후 H/W와 S/W간의 통신을 위해 BAC 모듈에서 이진산술부호화 하는데 필요한 값인 biari context 구조체의 변수인 count, state, MPS와 encoding environment pointer 구조체의 range, low, c, ebit_to_go와 symbol 값을 I/O 스트림을 통해 매 심벌마다 주고받을 수 있도록 설계 하였다.

BAC를 제외한 H.264/AVC의 부호기는 H.264/AVC의 참조 소프트웨어인 JM 버전을 이용하여 개발하였고, JM에서 사용되는 구조체변수를 그대로 이용하여 빠른 개발이 가능하게 하였다. 개발된 H.264/AVC 부호기의 경우 이진 심벌을 압축하기 위해 BAC 모듈에 데이터를 전송할 때 range, low, c, ebit_to_go, count, state, MPS, symbol 값을 I/O 스트림을 사용해서 주고받아야 하기 때문에 스트림에 사용되는 로직크기가 늘어나고, 전체 연산시간에도 영향을 주게 된다. 설계된 BAC 하드웨어 모듈은 기존에 발표된 구조[7]와 달리 하드에웨어 모듈에 I/O 스트림이 추가로 구현되어야 하기 때문에 FPGA로 구현했을 때, 로직의 사용량이 더 커진다.

2) CABAC H/W 설계 (방법2)

앞에서 제안한 설계방법1이 BAC부분만 하드웨어로 설계하였기 때문에 심벌과 그 심벌의 확률 업데이트 등을 소프트웨어 프로세스에서 전달받아야 하는 단점이 있어서, 두 번째 방법으로 그림12와 같이 M-coder에서 빈번한 액세스가 필요한 biari context type 구조체와 encoding environment pointer 구조체를 하드웨어 내에서 저장하도록 설계하여 구현하였다. 이와 같은 방법을 채택할 경우 M-coder에서 연산 하는데 필요한 값인 count, state, MPS와 range, low, c, ebit_to_go를 H/W메모리에 저장하여 연산 작업 후 H/W내의 메모리에서 곧바로 업데이트 하기 때문에 H/W의 메모리를 효과적으로 사용 가능하며 S/W와 H/W간의 I/O 스트림의 개수를 감소시킬 수 있는 장점이 있다. 과도한 스트림의 사용이 개선되고 복잡한 연산과 context업데이트 과정이 H/W에서 이루어지기 때문에 방법1의 H/W설계 방법에 비해 효율적이라 할 수 있겠다.

그림 13은 방법2에서 설계한 CABAC H/W의 구성도이다. 주요 블록으로 context모듈과 macro 모듈과 bitstream 출력부의 3부분으로 구성된다. 먼저, context 모듈은 정적메모리가 사용되며 다양한 신택스들의 컨텍스트 구성에 따른 실시간적으로 각종 변수들이 저장된다. 이와 같은 구조는 하드웨어를 효율적으로 사용하면서 입출력 스트림을 줄일 수 있는 효과적인 방법이다.

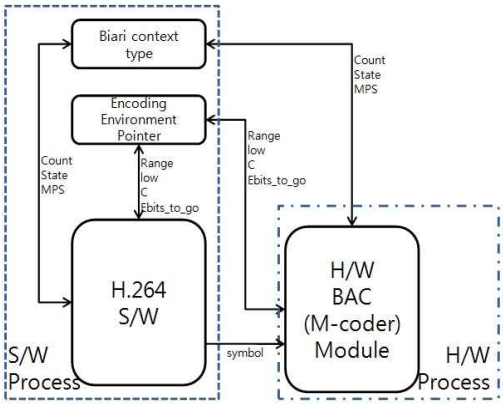

그림 11. BAC 하드웨어 모듈과 연동되는 H.264/AVC의 구조
Fig. 11. H.264/AVC structure co-operated with the BAC hardware module.

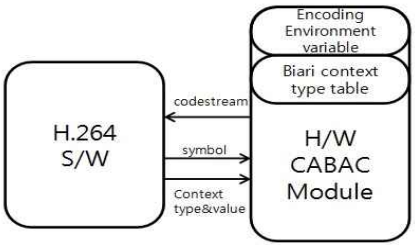

그림 12. CABAC 하드웨어 모듈과 연동되는 H.264/AVC의 구조
Fig. 12. H.264/AVC structure co-operated with CABAC hardware module.

소프트웨어 프로세스로부터 입력 스트림을 통해 context (type, value, symbol, mode) 값을 한꺼번에 입력받으면, (type, value)를 이용하여 macro 모듈에 저장되어 있는 (count, state, MPS) 값을 세팅한다. (count, state, MPS) 값들은 context (type, value)값에 의존하여 2차원적인 메모리 번지를 가져서 액세스 되도록 설계하였다. 마지막으로 (symbol, mode)를 이용하여 정규화 모드, 종결모드, 바이패스 모드, byte stuffing모드를 구분해 준 후에 해당되는 하드웨어 모듈이 동작된다.. 각각의 모드에서 비트스트림이 발생할 경우에 bitstream출력부를 통해 출력을 소프트웨어 프로세스에 기록하게 된다. 이 과정에서 macro 에 저장되어 있는 (count, state, MPS) 값들을 현재 값으로 업데이트하여 다시 메모리에 저장한다.

이러한 구성도를 통해 이루어지는 CABAC 압축 흐름도를 그림 14에 보였다. 연산에 필요한 context (type, value, symbol)을 넣어주면, (count, state, MPS)를 셋팅한다. 입력된 mode 값에 따라 4개의 이진압축모드가 동작되는 모듈로 심벌이 전달되어 압축을 수행한다. 수행 후에 다시 context의 (count, state, MPS) 값이 업데이트 된다. 과정 중에 비트스트림 출력이 발생하는 경우가 생기면 비트스트림 모듈을 통해 소프트웨어 프로세스 측에 비트 출력을 기록하게 된다.

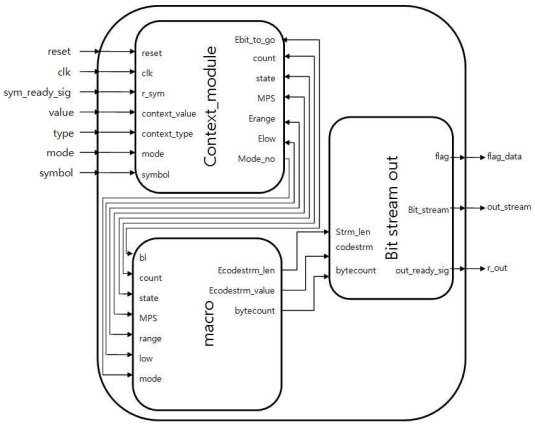

그림 13. 설계된 CABAC H/W의 구조
Fig. 13. Structure of designed CABAC H/W.

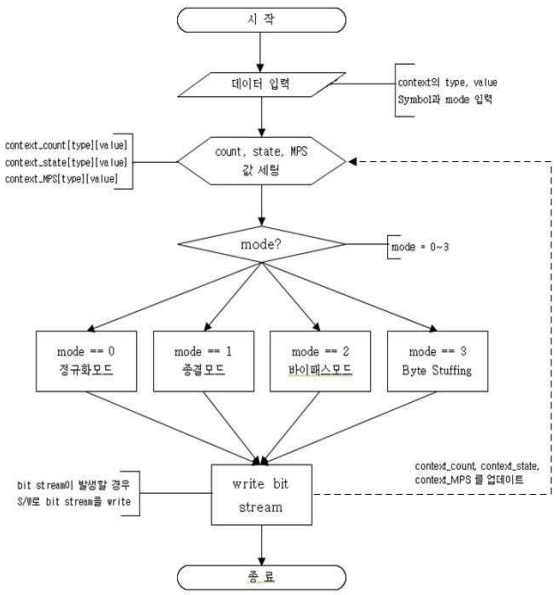

그림 14. CABAC의 H/W 흐름도
Fig. 14. Flowchart of CABAC hardware.

마지막으로 S/W와 H/W간의 입출력 스트림 설계에 따라서 하드웨어의 사용량과 연산속도는 큰 차이를 보인다. 따라서 본 논문에서는 최적의 스트림 설계를 진행하였다.

H/W로의 입력 데이터가 갖는 최대치를 모니터링 한 후 입력 데이터인 type, value, symbol, mode는 각각 8, 4, 2, 2bit의 최대 bit를 갖는 것을 알아내었다. 각각의 변수를 논리합으로 16bit의 입력 변수를 만들어 준 뒤 각각의 입력데이터가 1개의 스트림을 통해서 차례로 입력될 수 있도록 입력 스트림을 설계해 주었다. 그림 15는 간략하게 나타낸 입출력간의 통신 스트림의 구조를 보였다.

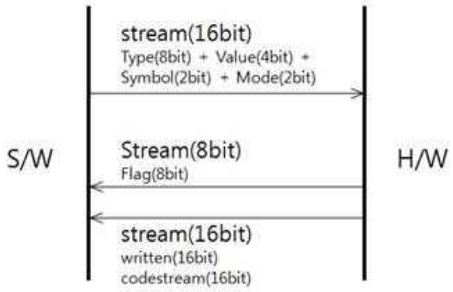

그림 15. 스트림 설계도
Fig. 15. Stream design.

출력 데이터인 codestream은 항상 출력되는 것이 아니라 출력 값이 존재함을 알려주는 변수인 flag가 0이 아닌 경우에만 출력을 발생 시키므로 항상 데이터가 발생하는 written value와 같은 스트림을 순차적으로 사용할 수 있도록 16비트 하나로 스트림을 설계하였다. 마지막으로 flag데이터는 실제 4비트 미만의 데이터를 갖고 있지만 ImpulseC에서 스트림은 4비트로 설계를 하여도 실제 FPGA에서는 8비트로 인식을 하도록 되어있기 때문에 8비트의 스트림 하나로 설계하였다.

Ⅳ. 실험 및 결과

4-1 기능 시뮬레이션

그림 15는 일정한 바이너리 값을 넣어주는 CABAC의 검증 프로그램을 만든 후 Modelsim SE 10.1c버전을 사용하여 M-coder의 하드웨어 시뮬레이션을 실시한 결과이다. 시뮬레이션 시 클록은 100 ps로 설정하였고 입력 바이너리 데이터는 총 32 byte의 테스트 시퀀스를 바이너리 값으로 변경하여 연속으로 입력되게 하였다. 그 결과 S/W 시뮬레이션에서 얻을 수 있는 결과 값인 'x1A' 가 정상적으로 출력함을 확인함으로써 하드웨어가 정상적으로 동작함을 확인 하였다.

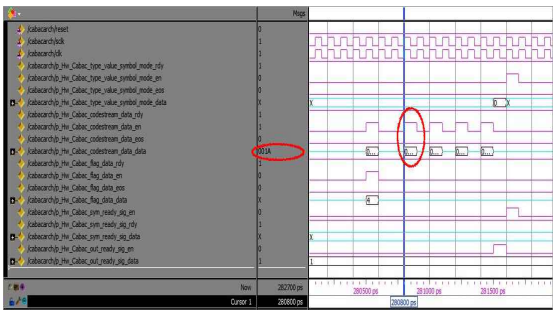

그림 16. CABAC의 기능 시뮬레이션 결과
Fig. 16. Simulation results of CABAC.

4-2 FPGA 사용량 비교

설계된 CABAC H/W와 동일하게 병행설계 기법으로 설계한 CAVLC (context adaptive variable length coding) H/W의 FPGA 사용량을 비교하여 표2에 보였다. CABAC 방법2와 CAVLC를 비교하면, slice와 F/F 및 LUT 모두 약 1/3로 감소된 것을 확인할 수 있었다. 그러나 IOB는 CAVLC에 비해 스트림이 더 많이 필요하여 2배 정도 증가하는 것을 알수 있었다.

또한 기존에 발표된 CABAC H/W설계 방식[17]의 slice사용량과 비교 하였을 때 약 2.7배, 또 다른 설계[18]과 비교하였을 때 약2.9배 늘어난 결과를 보였다. 그러나, 최근 FPGA 로직 용량이 급격히 늘어나고 있어서 그 차이는 중요도가 점점 줄어들고 있다. 본 논문에서 사용한 virtex-4 FX60 모델의 경우 slice가 26,624로 방법2로 설계한 경우에 percentage 사용량은 3.5%에 불과하여 기존의 설계보다 크게 나쁘지 않다. 하드웨어와 소프트웨어를 분리 개발하는 방식에 비해 설계 시간과 설계 비용을 감소 시킬 수 있는 이와 같은 하드웨어 설계 방법이 효과적이라 할 수 있겠다.

표 2. 설계된 CABAC H/W의 FPGA 사용량과 비교
Table 2. Amount of FPGA by the designed CABAC and comparison.

Logic Utilization	방법1	방법2	CAVLC [16]	[17]	[18]
Slices	1,493	925	3,484	341	318
Slice Flip Flops	627	578	1,658		121
4 input LUTs	1,658	1,626	5,776		603
bonded IOBs	146	84	40		
FIFO16/RAMB16s	3	7	8		
GCLKs	1	1	1		

Ⅴ. 결 론

본 논문에서는 하드웨어와 소프트웨어를 동시에 설계하는 병행설계(co-design)를 이용한 임베디드 시스템에서 사용 가능한 H.264/AVC의 CABAC 의 하드웨어 설계에 관한 것이다.

H/W-S/W 병행 설계 기법을 이용하여 H.264/AVC의 CABAC 모듈 중 이진 산술부호기와 문맥모델러부를 H/W로 설계하고, H.264/AVC 부호기의 나머지는 S/W로 동작하도록 설계하였다. CABAC 하드웨어 모듈에는 컨텍스트 모듈과 macro 모듈 및 비트스트림 모듈로 구성하였고, 컨텍스트 모듈과 macro 모듈은 정적메모리를 이용하여 구현하였다. 설계된 하드웨어 성능을 다른 논문에서 발표된 것들과 비교할 때, CAVLC에 비해서는 1/3 수준으로 감소시킬 수 있음을 확인하였다. 그러나. 다른 CABAC 방법에 비하여 2～3배 정도의 slice 사용량의 증가를 가져왔다. 그러나, 최근에 발매되는 FPGA의 성능을 생각하면 본 논문에서 설계한 방법에 의하더라도 전체 사용비율이 4% 이내인 것을 고려하면 이러한 비교결과는 크게 문제가 되지 않을 것으로 보인다. 본 논문에서 제안한 병행설계 방식의 장점으로 설계시간의 단축과 소프트웨어 엔지니어들이 하드웨어 가속기를 쉽게 설계할 수 있는 장점이 있다.

Full HD급 이상의 화질을 갖는 멀티미디어 기기의 보급과 함께 고압축, 고화질의 영상을 원활히 재생할 수 있는 하드웨어 코덱의 IP 확보를 위해 본 연구는 H.264/AVC 뿐만 아니라 동일한 CABAC엔트로피 코더를 사용하는 HEVC의 하드웨어 코덱을 제작하는데 있어서도 많은 도움을 제공할 것으로 생각한다.

감사의 글

이 논문은 2013년도 광운대학교 교내 학술연구비 지원에 의해 연구되었음.

참고문헌

[1] B. Bross, High efficiency video coding text specification draft 10 , JCT-VC of ISO/IEC and ITU-T, Geneva, Jan. 2003.

[2] JVT G050r1, Draft ITU-T recommendation and final draft international standard of joint video specification, ITU-T, 14496-10, May. 2003.

[3] D. G. Sim, Next generation video coding standard technique, IC Design Education Center Newsletter, pp. 04-09, July 2011.

[4] D. Marpe, H. Schwarz, and T. Wiegand, “Context-based adptive arithmetic coding in the H.264/AVC video compression standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, pp. 620-636, July 2003.

[5] D. Marpe and T. Wiegand, “A highly efficient multiplication-free binary arithmetic coder, and its application in video coding,” in *Proceedings IEEE International Conference on Image Processing 2003*, Barcelona: Spain, Sept. 2003.

[6] X. H.Tian, T. M.Le, X. Jiang, and Y. Lian, “A HW encoder with efficient context access scheme for H.264/AVC,” *IEEE International Symposium in Circuits and Systems*, pp. 18-21, May 2008.

[7] F. Zargari and E. Azimi, “An efficient hardware implementation for H.264 binary arithmetic encoder,” in *Proceedings of the 14th International Computer Society of Iran Computer Conference*, Tehran: Iran, pp. 105-109, 2009.

[8] K. Andra, T. Acharya, and C. Chakrabarti, “A multi-bit binary arithmetic coding technique,” in *2000 International*

Conference on Image Processing, Vancouver: Canada, pp. 928-931, Sept. 2000.

[9] W. B. Pennebaker, J. J.Mitchell, G. G. Langdon and R. B. Arps, "An overview of the basic principles of the q coder adaptive binary arithmetic coder," *IBM Journal of Research and Envelopment*, No. 6, pp. 717-726, Nov. 1988.

[10] X. Tian, *Entropy Coders of the H.264/AVC Standard*, Berlin Heidelberg, Germay: Springer-Verlag, 2011.

[11] C. Shinya, *H.264/AVC Textbook*, Seoul, Korea: Hongrung Publishing Company, 2005.

[12] Fraunhofer Heinrich-Hertz-Institut[Internet]. Available: http://www.hhi.fraunhofer.de/de/kompetenzfelder/image-processing/research-groups/image-video-coding/statistical-modeling-coding/fast-adaptive-binary-arithmetic-coding-m-coder.html

[13] P. S. Liu, J. W. Chen and Y.L. Lin, "A hardwired context-based adaptive binary arithmetic encoder for H.264 advanced video coding," *IEEE International Symposium on VLSI Design*, Vol. 1, pp. 1-4, Apr. 2007.

[14] Impulse Point and Sate Connect Manages BYOD[Internet]. Available: http://www.impulse.com

[15] D. Pellerin and S. Thibault, *Practical FPGA programming in C*, New Jersey, United States: Prentice Hall, 2005.

[16] D. S. Wang, Hardware implementation of DCT and CAVLC for H.264/AVC based on co-design, M.S. thesis, Kwangwoon University, Seoul, Dec. 2010.

[17] R. R. Osorio and J. D. Bruguera, "A new architecture for fast arithmetic coding in H.264 advanced video coder," in *8th Euromicro Conference on Digital System Design*, Porto: Portugal, pp. 298-305, Sept. 2005.

[18] J. L. Nunez-Yanez, V. A. Chouliaras, D. Alfonso, and F.S.Rovati, "Hardware assisted rate distortion optimization with embedded cabac accelerator for the H.264 advanced video codec," *IEEE Transactions on Consumer Electronics*, Vol. 52, No. 2, pp. 590-597, 2006.

조 영 주 (Young-Ju Cho)
2007년 2월 : 광운대학교 전자통신공학과(공학사)
2014년 8월 : 광운대학교 전자통신공학과 (공학석사)
※관심분야 : 영상처리, HEVC, H.264, H/W-S/W co-design

고 형 화 (Hyung-Hwa Ko)
1979년 2월 : 서울대학교 전자공학과(공학사)
1982년 2월 : 서울대학교 전자공학과(공학석사)
1989년 2월 : 서울대학교 전자공학과(공학박사)
1985년 3월~현재 : 광운대학교 전자통신공학과 교수
※관심분야 : HEVC, H.264, H/W-S/W co-design, JBIG2, Watermarking