2014년 5월 전자공학회 논문지 제 51권 제 5 호
Journal of The Institute of Electronics and Information Engineers Vol. 51, NO. 5, May 2014 http://dx.doi.org/10.5573/ieie.2014.51.5.114

논문 2014-51-5-14

HEVC 복호화기의 메모리 접근 복잡도 분석

(An Analysis of Memory Access Complexity for HEVC Decoder)

조 송 현*, 김 영 남*, 송 용 호**

(Song Hyun Jo, Youngnam Kim, and Yong Ho Song©)

요 약

HEVC는 JCT-VC에 의해 개발된 최신 비디오 코딩 표준이다. HEVC는 H.264/AVC에 비해 약 2배의 주관적 코딩효율을 제공한다. HEVC 개발의 주요목표 중 하나는 UHD급 비디오를 효율적으로 코딩하는 것이기 때문에, HEVC는 UHD급 비디오를 코딩하는데 널리 사용될 것으로 예측된다. 이러한 고해상도 비디오의 복호화는 많은 양의 메모리 접근을 발생시키기 때문에 복호화 시스템은 고대역폭의 메모리 시스템 및 내부 통신 아키텍처가 필요하다. 이러한 요구사항을 파악하기 위해서 본 논문은 HEVC 복호화기의 메모리 접근 복잡도를 분석한다. 우리는 먼저 임베디드 프로세서와 데스크탑에서 동작하는 소프트웨어 HEVC 복호화기의 메모리 접근량을 측정하였다. 또한 우리는 HEVC 복호화기의 데이터흐름을 분석하여 HEVC 복호화기의 메모리 대역폭 모델을 만들었다. 측정결과, 소프트웨어 복호화기는 6.9~40.5GB/s의 DRAM 접근을 하였다. 또한 분석결과에 따르면 하드웨어 복호화기는 2.4GB/s의 DRAM 대역폭을 요구하는 것으로 파악된다.

Abstract

HEVC is a state-of-the-art video coding standard developed by JCT-VC. HEVC provides about 2 times higher subjective coding efficiency than H.264/AVC. One of the main goal of HEVC development is to efficiently coding UHD resolution video so that HEVC is expected to be widely used for coding UHD resolution video. Decoding such high resolution video generates a large number of memory accesses, so a decoding system needs high-bandwidth for memory system and/or internal communication architecture. In order to determine such requirements, this paper presents an analysis of the memory access complexity for HEVC decoder. we first estimate the amount of memory access performed by software HEVC decoder on an embedded system and a desktop computer. Then, we present the memory bandwidth models for HEVC decoder by analyzing the data flow of HEVC decoding tools. Experimental results show the software decoder produce 6.9-40.5 GB/s of DRAM accesses. also, the analysis reveals the hardware decoder requires 2.4 GB/s of DRAM bandwidth.

Keywords : HEVC, decoder, complexity, memory access, DRAM bandwidth

Ⅰ. 서 론

HEVC[1~2]는 JCT-VC(the Joint Collaborative Team on Video Coding)가 개발한 최신 비디오 코덱 표준이다. HEVC main profile 코덱은 기존에 널리 쓰이던 비디오 코덱인 H.264/AVC baseline profile에 비해 동일 bit-rate에서 주관적 화질을 기준으로 코딩효율이 2배정도 향상되었다[3]. 이에 따라, H.264/AVC를 사용 시

* 학생회원, ** 정회원, 한양대학교 전자컴퓨터통신공학과
(Department of Electronics and Computer Engineering, Hanyang University)
※ 본 연구는 미래창조과학부 및 정보통신산업진흥원의 시스템반도체 설계인력양성사업의 연구결과로 수행되었음.(NIPA-2014-H0601-14-1001)
© Corresponding Author(E-mail: yhsong@hanyang.ac.kr)
접수일자: 2014년4월 5일, 수정일자: 2014년4월10일
수정완료: 2014년4월29일

1024p HD급 영상 전송이 가능한 대역폭을 지닌 채널의 경우, HEVC를 사용하면 코딩효율의 향상으로 4K급의 영상 전송이 가능하다는 발표[4]도 있다.

HEVC는 특히 4K/8K UHD급의 고해상도 비디오 압축에 널리 쓰일 것으로 기대되고 있다. HEVC 설계의 주요 목표 중 하나는 이러한 고해상도 비디오에 대한 코딩 효율을 높이는 것이다. 이를 위해서 예측, 양자화, 그리고 변환을 수행하는 화소 범위의 최대 크기가 H.264/AVC에서의 16x16에서 64x64로 증가하였고, 관련 코딩 도구들의 알고리즘이 고해상도 영상에서 좋은 코딩효율을 달성하도록 설계되었다. 고해상도 비디오 기반 제품의 공급과 수요가 점차 증가함에 따라 HEVC가 관련 산업과 소비자에게 의미 있는 기여를 할 것으로 기대된다.

고해상도 비디오를 복호화 하는 작업은 저해상도 비디오의 경우에 비해 대량의 비디오 데이터를 처리한다. 이 때, 시스템 내부에 대량의 데이터 트래픽이 발생하고, 메모리의 대역폭에 대한 요구사항이 증가하게 된다. 따라서 HEVC 복호화를 수행할 시스템은 캐시, DRAM 등의 메모리 시스템과 버스, NoC(Network-on-a-chip)와 같은 내부 통신 아키텍처가 이러한 요구사항을 만족하도록 설계되어야 한다. 그러므로 HEVC 복호화기의 메모리 접근 특성에 대한 분석이 필요하다. 그런데 기존의 HEVC 복잡도 분석 논문들은 대부분 수행시간 및 계산복잡도를 대상으로 연구를 수행하였다[5~7].

따라서 본 논문에서는 HEVC 복호화기의 메모리 접근 특성을 분석한다. 우리는 HEVC 복호화기의 메모리 접근 복잡도를 두 가지 관점에서 파악하였다. 첫 번째는 소프트웨어 HEVC 복호화기의 메모리 접근량을 실제 시스템 혹은 시뮬레이터로 측정하는 하는 방법이고, 두 번째는 하드웨어 HEVC 복호화기의 메모리 접근량을 HEVC 복호화기의 도구 수준 데이터 흐름을 이론적으로 분석함으로써 계산하는 방법이다.

먼저 소프트웨어 복호화기의 메모리 접근량을 측정하는 하는 방법은 임베디드 프로세서에 대해서는 시뮬레이터를 이용하여 간접적으로, 데스크탑 프로세서에 대해서는 실제 시스템을 통해 직접적으로 메모리 접근량을 측정하였다. 소프트웨어 복호화기의 경우 주 관심사 중 하나는 캐시 아키텍처 및 크기에 따라 데이터 접근량이 어떻게 변화하느냐이다. 이를 파악하기 위하여 우리는 캐시 크기를 16KB부터 3072KB까지 변화시켜가며 캐시 메모리, DRAM 접근량, 캐시 적중률 등을 파악하였다.

두 번째는 HEVC 복호화기의 도구 수준 데이터 흐름을 이론적으로 분석하여 하드웨어 HEVC 복호화기의 데이터 접근 특성을 파악하는 방법이다. 앞서 설명한 소프트웨어 복호화기의 메모리 접근 특성 측정 결과는 하드웨어 HEVC 복호화기 관점에서는 불필요한 DRAM 접근을 포함하고 있다. 이 불필요한 메모리 접근은 소프트웨어 복호화기 자체의 비효율적인 데이터 접근 방식, 캐시의 정책의 비효율성, 제한된 캐시 크기 등에 의해 발생한다. 반면 HEVC 복호화기의 도구 간 데이터 흐름을 이론적으로 분석하면 불필요한 메모리 접근이 발생하지 않도록 효율적으로 설계한 하드웨어 복호화기가 발생시키는 메모리 접근량에 대한 분석이 가능하다. 우리는 먼저 전체 도구 간 데이터 흐름을 파악하고, 이중 DRAM 메모리 접근을 발생시키는 데이터 흐름을 선별하였다. 이후, 선별된 데이터 흐름에 대해서 대역폭을 모델링하여 하드웨어 HEVC 복호화기의 메모리 접근량을 이론적으로 계산하였다.

측정결과, 소프트웨어 복호화기는 6.9GB/s에서 40.5GB/s의 DRAM 접근을 하였다. 또한 분석결과에 따르면 하드웨어 복호화기는 2.4GB/s의 DRAM 대역폭을 요구하는 것으로 파악된다.

본 논문의 Ⅱ장에서는 HEVC 복호화기의 복잡도를 다룬 관련연구를 살펴본다. Ⅲ장에서는 HEVC 복호화기를 설명한다. 이후 Ⅳ장에서는 소프트웨어 HEVC 복호화기의 메모리 접근량을 측정 및 분석한다. Ⅴ장에서는 HEVC 복호화기의 데이터 흐름을 분석하여 하드웨어 HEVC 디코더의 메모리 접근량을 계산한다. Ⅵ장에서는 결론을 맺는다.

Ⅱ. 관련연구

HEVC 복호화기의 복잡도에 대해 기존에 몇 가지 연구가 진행되어왔는데, 대부분 복호화수행시간이나 계산복잡도에 초점을 두고 있다. Viitanen[5]와 복수의 기고서[6~7]는 복호화 수행시간에 대한 분석을 수행하였다. 특히, Viitanen[5]은 매우 다양한 bitstream에 대해 복호화 수행시간을 측정하였다. Ahn[6]은 복호화 도구들의 연산 복잡도를 이론적으로 자세히 분석하였다. Bossen[7]은 복호화 도구 별 주요 알고리즘의 계산복잡

도를 H.264/AVC와 비교하였다. 또한 HEVC 참조 소프트웨어인 HM 복호화기의 도구별 수행시간을 데스크탑 환경에서 비교하였고, HM에 비해 수행시간 측면에서 최적화된 소프트웨어 HEVC 복호화기의 도구별 수행시간을 임베디드 환경에서 측정하였다. 그러나 본 논문과 달리 위 연구결과들은 공통적으로 메모리 접근 측면에서의 복잡도 분석이 미미하다.

Ⅲ. HEVC 복호화기

1. 자료구조

HEVC는 하나의 영상을 복수의 GOP(group of pictures)로 구성하였으며, 각 GOP는 복수의 프레임을 보유하고 있다. 각 프레임은 그림 1과 같이 하나 혹은 여러 개의 slice/tile로 분리된 후 다시 CTU(coding tree unit)이라는 정사각형 자료구조로 분리된다. 하나의 CTU는 Quad-tree 형태로 분리되며, 말단 노드의 자료구조는 CU(coding unit)라고 칭한다[8]. CU는 intra-coded CU, inter-coded CU, skipped CU의 세 가지 형태가 있다. 마지막으로 CU는 복수의 PU로 분리되는데, 오직 inter-coded CU안의 PU만이 MC에 의해 처리된다.

각 화소는 luma 샘플과 blue chroma 샘플, red chroma 샘플, 이렇게 세 개의 색체요소로 구성된다. HEVC main profile에서는 화소를 표현하는 데 사용되는 비트수를 감소시키기 위해 네 개의 화소가 red와 blue chroma 샘플을 공유한다. 반면 각 화소는 하나의 luma 샘플을 보유한다.

CTU, CU, PU는 해당영역의 화소와 관련 syntax elements로 정의된다. 반면 CTB(coding tree block), CB(coding block), PB(prediction block)와 같은 block은 unit과 같은 영역내의 특성 색체요소의 샘플 값만을 가리키는 용어이다. 따라서 한 unit은 luma block, blue chroma block, red chroma block의 세가지 블록을 보유한다.

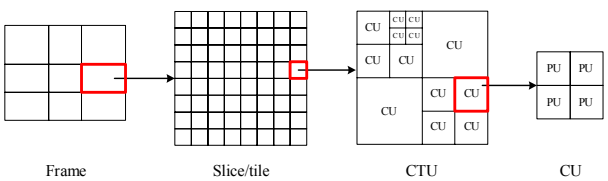

그림 1. HEVC 자료구조
Fig. 1. The data structure of HEVC.

2. Motion compensation

HEVC에는 여러 가지 복호화 도구가 있는데, 그 중 연산량과 데이터 접근이 가장 많은 도구 하나가 MC(motion compensation)[9]이다. MC는 inter-coded CU내 PU의 화소 값을 예측한다. MC는 이전에 복호화된 프레임 내에서 유사한 화소값을 갖는 블록을 참조하여 예측한다. 이러한 작업은 현재 화면이 이전 화면과 유사함에 따라 갖는 시간적 중복성을 제거하여 부호화 효율을 향상시키기 때문에 수행한다. HEVC는 MC 수행을 위해서 이전에 복호화 된 영상의 일부를 DPB(decoded picture buffer)에 저장한다. 예측을 위해 참조하는 화소 값으로 이전 프레임의 샘플 값을 그대로 사용하는 경우도 있고, 정밀도를 향상시키기 위하여 보간(interpolation)을 통해 샘플 사이의 fractional 샘플 값을 생성하여 사용하는 경우도 있다. 보간에는 1-D, 2-D interpolation이 있는데, 2-D가 1-D보다 더 많은 데이터를 참조한다.

Ⅲ. 메모리 접근량 측정 및 분석

이번 장에서는 소프트웨어 HEVC 복호화기 의 메모리 접근량을 임베디드 프로세서는 시뮬레이션을 통해, 데스크탑 PC용 프로세서는 실측을 통해 분석한다. 먼저 측정 시 가정한 하드웨어 및 소프트웨어 환경과 측정방법을 설명하고 메모리 접근량 측정결과를 설명한다.

1. 실험환경

임베디드 프로세서 기반 실험은 Xtensa LX4 processor 기반 시스템을 모델링 한 ISS(instruction set simulator)[10]을 이용하여 진행 되었다. 시뮬레이션의 속도를 향상시키기 위하여, 실험은 ISS에서 제공하는 혼합 시뮬레이션을 이용하여 수행하였다. Xtensa 프로세서의 ISS에는 크게 두 가지 모드가 있다. 하나는 기능적 시뮬레이션(functional simulation)인데, 이 방법은 기능적인 동작만 시뮬레이션하기 때문에 시뮬레이션 속도가 빠르지만 시간 관련 성능 분석이 어렵다. 나머지 하나는 사이클 정확도 시뮬레이션(cycle-accurate simulation)인데, 이 방법은 수행시간을 사이클 단위로 시뮬레이션하기 때문에 성능분석이 가능하나 시뮬레이

션 속도가 느리다. 우리는 두 가지 시뮬레이션 방법을 혼합한 방법을 사용한다. 즉, 대부분의 명령어는 기능적 시뮬레이션을 적용하되 일부 명령어는 사이클 정확도 시뮬레이션을 적용하였다. 구체적으로는, 시뮬레이션 되는 명령어의 개수를 기준으로 기능적 시뮬레이션과 사이클 정확도 시뮬레이션을 100:1의 비율로 번갈아 가며 수행하였다. 이 때 기능적 시뮬레이션이 연속으로 처리하는 명령어의 개수는 100,000,000개이고, 사이클 정확도 시뮬레이션이 연속으로 처리하는 명령어의 개수는 1,000,000개이다. 성능 측정은 사이클 정확도 시뮬레이션으로 처리된 명령어를 기준으로 수행된다. 메모리 접근량은 사이클 시뮬레이션으로 측정된 메모리 접근량의 101배로 크기를 조정하였다. 전체 명령어 개수 대비 사이클 정확도 시뮬레이션된 명령어의 비율이 101:1이기 때문이다.

데스크탑 PC용 프로세서 기반 실험은 Intel Core 2 프로세서에서 실측을 통해 수행되었다. 이 측정은 프로세서 내부에 존재하는 성능계수기(performance counter)를 사용하는 VTune 프로파일러를 이용하여 수행되었다.

실험 대상이 되는 캐시 아키텍처의 사양은 표 1과 같다. Xtensa LX4는 configurable processor이기 때문에 시스템 사항을 자유롭게 설정할 수 있다. 따라서 L1 캐시의 경우 16KB에서 64KB까지 다양한 크기를 설정하여 실험을 수행하였다. Xtensa LX4는 단일 레벨 캐시를 사용한다. Intel Core 2 프로세서는 코어 하나를 기준으로 64KB 크기의 L1 캐시와 3072KB 크기의 L2 캐시를 보유하고 있다.

표 1. 실험 대상 캐시 아키텍처
Table 1. The specification of caches experimented.

캐시	항목	Tensilica Xtensa LX4	Intel Core 2
L1	용량	D-캐시: 16~64KB I-캐시: 32KB	D-캐시: 32KB I-캐시: 32KB
	세트 연관도	2-way 또는 4-way	8-way
	Line 크기	64-byte	64-byte
L2	용량	-	3072KB
	세트 연관도		12-way
	Line 크기		64-byte

실험용 HEVC 디코더는 표준 reference HEVC decoder인 HM 6.0이다. 디코더의 입력 bitstream으로는 2560x1600해상도, 30 FPS(frame per second), 그리고 총 150 frames인 *Traffic*과 *PeopleOnStreet* 영상[11]을 인코딩한 bitstream이 사용되었다. 이 bitstream은 HM 6.0 부호화기를 사용하여 random access 모드에서 양자화 계수 32와 37로 각각 인코딩되었다. 좀 더 최신의 HM은 HM 6.0 버전과 메모리 접근량이 다를 수도 있다. 하지만 QP37로 부호화한 *Traffic* bitstream을 Intel Core 2 프로세서에서 HM 6.0과 10.0으로 복호화하고 메모리 접근량을 비교해보니 세부 항목별로 5%이하의 차이만이 존재하였다. 따라서 HM 6.0에서의 메모리 접근량 실험 결과가 최신 HM에서도 유효할 것으로 기대된다.

2. 복호화 수행 시간

실험에 사용된 bitstream의 특성을 알아보기 위하여 그림 2와 같이 각 bitstream의 복호화 수행시간을 복호화 도구 단위로 측정하였다. 이 때, 데이터 캐시의 크기는 32KB로 설정하였다. *PeopleOnStreet*가 *Traffic*보다 복호화에 더 많은 시간이 소요되었다. 또한 QP값이 작을수록 복호화 소요시간이 증가하였는데 QP값이 작을수록 데이터 양이 증가함을 감안하면 당연한 현상이다. 복호화 도구별로는 상이한 경향이 발견되었다. Entropy coding, 역변환/양자화, deblocking filter, 그리고 intra prediction는 전체 수행시간의 경향과 마찬가지로 *Traffic*보다는 *PeopleOnStreet*에서 그리고 낮은 QP값에서 더 오랜 수행시간이 소요되었다. 반면 Inter prediction은 모든 bitstream에서 수행시간의 변화가 크기 않았다. SAO는 특이하게도 *Traffic*의 경우 높은 QP값에서 더 오랜 수행시간이 소요되었다.

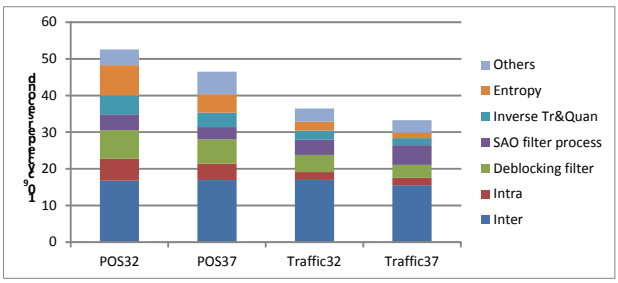

그림 2. 복호화 도구 별 수행시간
Fig. 2. The execution time for coding tools.

3. 메모리 접근량

HEVC 디코더의 메모리 접근량은 캐시 메모리로의 접근량과 DRAM 접근량으로 나누어 측정되었다. DRAM 접근량은 캐시 메모리 접근량에 비해 시스템 성능에 좀 더 직접적인 영향을 끼친다. 왜냐하면 DRAM 접근은 공유자원인 시스템 버스에 트래픽을 발생시켜 시스템 성능 저하를 일으킬 가능성이 있기 때문이다. 반면 캐시 접근은 일반적으로 프로세서와 캐시 사이의 전용 통신채널을 통해 수행되므로 시스템 성능에 직접적인 영향을 끼치지 않는다. 하지만 캐시 접근은 시스템 성능에 간접적으로 영향을 끼친다. 이는 대부분의 DRAM 접근은 캐시 누락(miss)에 의한 DRAM 읽기, write-back 동작에 의한 DRAM 쓰기에 의해 발생하기 때문이다. 캐시 접근량은 DRAM 메모리 액세스량의 이론적 최대치로 간주 될 수 있다. 캐시나 기타 지역 메모리(local memory)가 존재하지 않는 경우, 캐시 접근량은 데이터 캐시 접근량과 동일할 것이기 때문이다. 참고로 데이터 캐시 접근량은 프로그램 수행 중 load, store 명령어가 접근하는 데이터 양과 동일하다.

가. 캐시 접근량

캐시 접근량을 측정한 결과는 그림 3과 같다. QP값이 높을수록 캐시 접근량이 감소한다. 이는 QP값이 높을수록 색체를 표현하는데 사용되는 데이터크기가 줄어들기 때문이다. 캐시 접근은 명령어 인출, 데이터 load, 데이터 store 순으로 많이 발생하였다. 모든 명령어 실행마다 명령어 인출이 필요하므로 명령어 인출로 인한 명령어 캐시 접근이 가장 많다. Load, store 중 load의 양이 더 많은 것은 역양자화를 제외한 나머지 대부분의 코딩 도구들이 결과물로 생성되는 데이터보다 계산에 사용되는 데이터의 양이 많기 때문인 것으로 보인다. 예를 들어 가장 많은 수행시간이 소요되는 MC의 경우 1~2개의 참조 블록 화소값과 움직임 벡터 등이 입력데이터로 사용된다. 반면 그 결과물은 1개의 현재 블록 화소 값에 불과하여 그 크기가 상대적으로 작다.

나. DRAM 접근량

HEVC 디코더 수행 시 발생하는 DRAM접근량은 그림 4와 같다. 이 실험결과는 2-way set associative인 16KB 데이터 캐시를 가정하였다. 우선 명령어 인출이 캐시 접근량에 비해 대폭 감소하였다. 이는 명령어 인출이 자주 발생하므로 캐시 메모리 접근 횟수는 많지만, 캐시 적중률이 데이터 load, store에 비해 매우 높아 DRAM으로의 접근량은 상대적으로 매우 적기 때문이다. DRAM 접근에 있어 데이터 읽기 및 쓰기(read and write)는 캐시에서 동작하는 데이터 load, store와 1:1 대응되지 않으므로 직접 비교하지 않음을 유의해야 한다. 예를 들어 캐시 블록이 오랫동안 사용되지 않아 캐시에서 DRAM으로 축출 시 DRAM관점에서는 data write가 발생하게 된다. 이 때 데이터 쓰기는 load나 store 명령과는 직접적인 대응이 되지 않는다.

캐시 접근 (그림 3) 중 load와 store 명령에 의한 접근은 캐시누락 시 DRAM으로의 데이터 읽기 접근을 발생시킨다. 이 때 DRAM에서의 읽기연산은 load 및 store 명령어가 필요로 하는 데이터에 대해서만 수행되는 것이 아니라 해당 데이터가 포함된 캐시 블록 전체에 대해서 수행된다는 점을 유의하여야 한다. 예를 들어 그림 4의 데이터 읽기 접근량을 구하려면 load 및

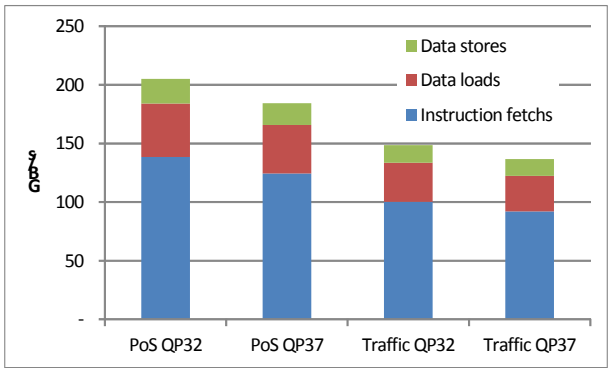

그림 3. 데이터 및 명령어 캐시 접근량
Fig. 3. The amount of data and instruction cache access.

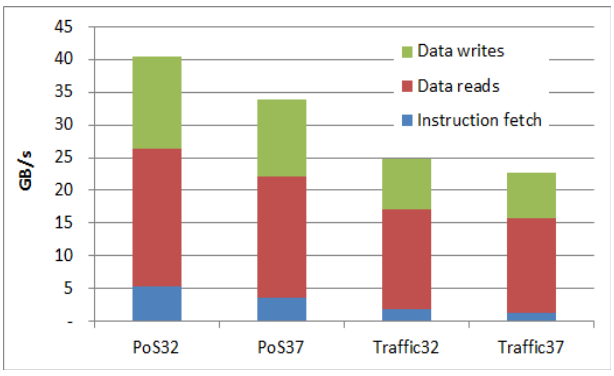

그림 4. Bitstream별 DRAM 접근량
Fig. 4. The amount of DRAM access for test bitstreams.

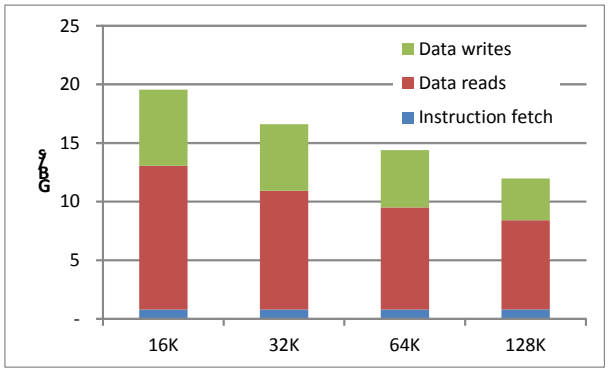

그림 5. 캐시 크기별 DRAM 접근량
Fig. 5. The amount of DRAM access with various size of caches.

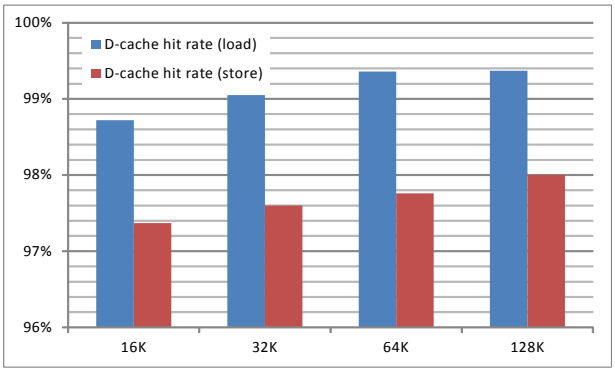

그림 6. 데이터 캐시 크기별 적중률
Fig. 6. Cache hit ratio for various size of data cache.

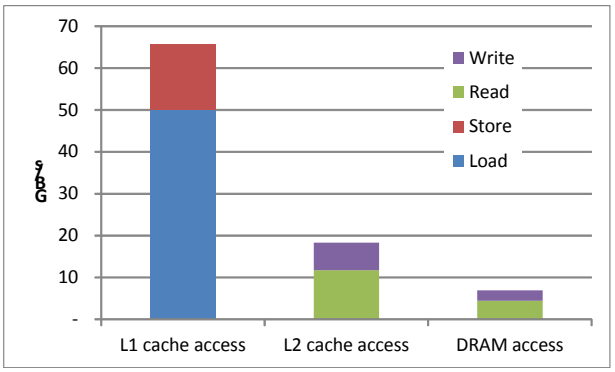

그림 7. Intel Core 2 프로세서에서의 메모리 접근량
Fig. 7. The amount of memory access caused by the Intel Core 2 processor.

store 명령에 의한 캐시 접근량에 캐시 누락률(1 – 캐시 적중률)을 곱할 뿐만 아니라 'load 및 store 명령어가 요구하는 데이터 양(Xtensa에서는 4Bytes) 대비 캐시 블록 크기(Xtensa에서는 64Bytes)'인 16을 곱하여야 한다.

그림 5는 데이터 캐시 크기에 따른 DRAM 접근량을 측정한 결과를 나타낸다. DRAM 접근량은 데이터 캐시 적중률이 높아질수록 줄어든다. 캐시 적중률은 캐시의 다양한 특성 중 특히 크기에 비례하는 경향이 강하므로 그 효과를 분석하기 위해 본 실험을 수행하였다.

우리는 캐시 크기를 16KB부터 128KB까지 변화시켜가며 DRAM 접근량을 측정하였다. 입력으로는 *Traffic* 영상을 QP37로 부호화한 bitstream을 사용하였다. 16KB일 때의 접근량이 그림 4의 결과와 상이한데 이는 본 실험에서 시뮬레이션한 캐시의 set assocativity가 4-way로 증가했기 때문이다. 데이터 캐시 크기가 증가할수록 DRAM 접근량이 감소하였다. 명령어 캐시는 크기가 32KB로 고정하였으므로 명령어 인출에 의한 접근량은 변화가 없었다. 실험에서 사용한 가장 큰 데이터 캐시 크기인 128KB까지는 DRAM 접근량이 감소가 포화(saturation)되지 않았기 때문에 더 큰 크기의 캐시 사용도 유의미 할 수 있을 것으로 판단된다.

그림 6는 데이터 캐시 크기에 따른 캐시 적중률을 나타낸다. 캐시 아키텍처는 캐시 크기별 DRAM 접근량 실험과 동일하다. Load와 store로 구분하여 적중률을 측정하였다. 전체적으로 데이터 캐시의 크기가 커질수록 캐시 적중률이 높아지는 특성을 보인다. 단, 64KB와 128KB 데이터 캐시는 load 적중률에 있어 유의미한 차이가 없다. 따라서 64KB 데이터캐시에서 load 적중률이 포화된 것으로 판단되며, 이에 따라 더 큰 데이터 캐시를 쓰더라도 load 적중률이 향상되지 않을 것으로 예측된다. 명령어 캐시는 32KB 크기에서 99.94%의 높은 적중률을 보이며 전체 DRAM 접근량에서 명령어 읽기가 차지하는 비중이 적기 때문에 캐시 사이즈를 증가시킬 필요성이 적다.

우리는 32K보다 더 큰 캐시를 사용했을 시의 메모리 접근량을 파악하기 위하여, Intel Core 2 프로세서에서의 메모리 접근량을 그림 7과 같이 조사하였다. 주요 관찰사항은 다음과 같다. (1) Xtensa 프로세서에서의 캐시 접근량은 Core 2 프로세서에서 L1 캐시 접근량에 대응된다. 두 가지 모두 load, store 명령 및 명령어 인출에 의해 접근하는 데이터 양을 나타내기 때문이다. Xtensa에 비해 Core 2의 경우 load, store 명령어의 수와 전체 수행 명령어 수가 모두 더 많은 것으로 나타났다. 하지만 해당 명령어간의 상대적인 비율은 유사하였다. (2) Xtensa에서 32KB 캐시를 사용하는 경우의 DRAM 접근량은 Core 2 프로세서에서 L2 캐시 접근량에 대응된다. Core 2 프로세서의 L1 캐시의 크기가

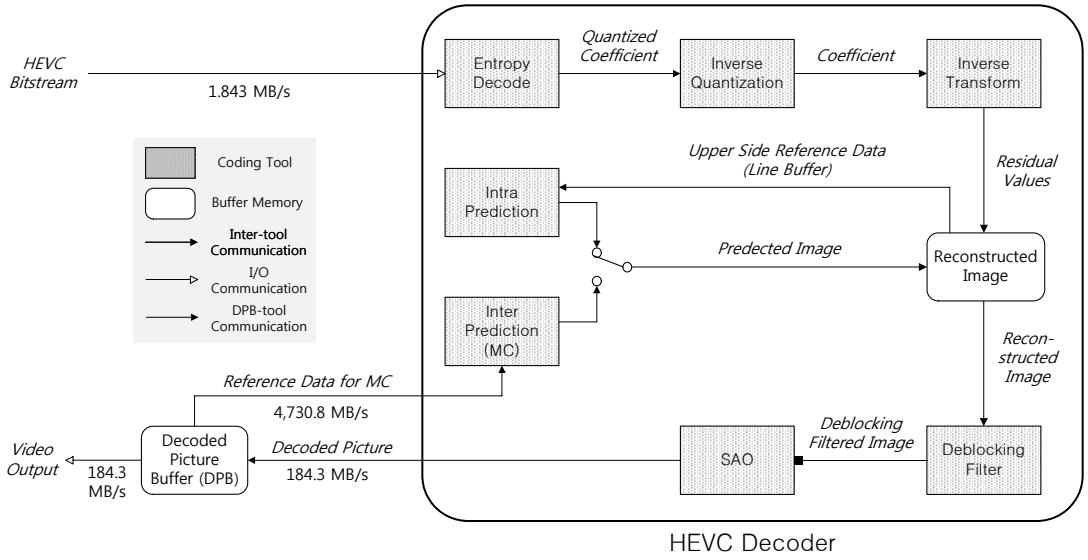

그림 8. HEVC 복호화기의 데이터 흐름.
Fig. 8. Data flow of HEVC decoder.

32KB이므로 두 가지 모두 32KB의 캐시가 하위 메모리에 읽기 및 쓰기 하는 데이터 양을 나타내기 때문이다. 이번에도 Core 2 프로세서가 조금 더 많은 데이터 접근량을 보이면서 읽기와 쓰기의 비율은 유사하였다. (3) Core 2 프로세서의 DRAM 접근은 L1캐시와 함께 3072KB크기의 L2 cache까지 사용하는 상황에서 벌어진다. 이 경우 읽기와 쓰기를 합쳐서 6.94GB/s의 데이터 접근이 발생하였다. 이는 32KB 캐시를 사용한 L2 캐시 접근량의 37.9%에 해당하는 량이다. 다만, 캐시 크기 증가에 다른 데이터 접근량 감소가 포화(saturation)되는 캐시 크기는 본 논문의 실험결과로는 단정하기 힘들다. 즉, 64KB와 3072KB의 중간 지점에서 포화되었을 수도 있고 3072KB보다 더 큰 크기에서 포화될 수도 있다.

Ⅳ. 도구 수준 데이터 흐름 분석을 이용한 하드웨어 HEVC 복호화기의 DRAM 접근량 분석

이번 장에서는 ITC를 기반으로 HEVC 복호화기의 데이터 접근량을 분석한다. 먼저 도구 수준 데이터 흐름을 분석 한 후, 이를 이용하여 하드웨어로 구현된 복호화기의 DRAM 접근량을 분석한다. 이후 하드웨어 복호화기의 DRAM 접근량과 Ⅲ장에서 측정한 소프트웨어 HEVC 복호화기의 DRAM 접근량을 비교 분석한다.

1. HEVC 도구 수준 데이터 흐름

HEVC 복호화기의 도구 수준 데이터 흐름은 그림 8과 같다. 색깔이 있는 사각형은 복호화 도구, 색깔이 없는 사각형은 버퍼, 화살표는 데이터 흐름을 나타낸다. 이동하는 데이터의 이름은 이탤릭체로 명시되어 있다. 복호화기 내부의 버퍼는 생략되었다. 이 데이터 흐름을 크게 입출력(I/O communication), DPB와 HEVC 복호화기 간의 통신(DPB-tool communication), HEVC복호화기 내부에서의 도구 간 통신(inter-tool communication)의 세 가지 종류로 나눌 수 있다. I/O communication은 입력인 *HEVC bitstream*와 출력인 *Video output*이 있다. 이 중 입력인 *HEVC bitstream*은 HEVC 복호화기로 직접 입력된다. 반면 *Video output*은 HEVC 복호화기 에서 화면으로 바로 이동하지 않고 외부 DRAM에 위치한 DPB에 저장된 후 이동된다. 버퍼를 이용하여 이동하는 이유는 복호화 된 영상중 일부를 이후에 MC가 사용하기 때문이다. 또한 이 버퍼가 외부 DRAM에 위치하는 이유는 SRAM등으로 구현되는 하드웨어 내부 버퍼에 저장하기에는 용량이 크기 때문이다. DPB-tool communication에는 *Decoded picture*와 *Reference data for MC*가 있다. SAO가 필터링을 수행하여 복호화가 완료된 *Decoded picture*는 DPB에 저장된다. 이후 이 영상을 MC가 참조하면 *Reference data for MC*가 DPB에서 MC로 이동한다.

마지막으로 HEVC 복호화기 내부에서는 *Coefficient*, *Residual value* 등의 다양한 inter-tool communication 이 발생한다. 복호화기 내부 데이터 흐름은 VLSI등 하드웨어로 HEVC 복호화기를 구현하는 경우에 시스템 버스에 데이터 트래픽을 발생시키지 않는 것이 일반적이다. 이는 HEVC 복호화기에 데이터 흐름에 최적화된 버퍼가 존재하여 데이터를 복호화기 내부에서만 보존하기 때문에 가능하다.

2. 하드웨어 HEVC 복호화기의 메모리 접근량

앞서 분석한 데이터 흐름 중 하드웨어 HEVC 복호화기의 메모리 접근을 발생시키는 데이터 흐름은 *HEVC bitstream*, *Decoded picture*, *Reference data for MC*이다. 여기서는 HEVC 복호화기가 데이터 흐름에 적합하게 설계된 버퍼를 보유하여 복호화기 내부의 데이터가 외부 DRAM에 버퍼링되지 않는다고 가정하였다. 이 경우, *HEVC bitstream*, *Decoded picture*, *reference data for MC*의 데이터 대역폭을 계산하여 합하면 하드웨어 HEVC 복호화기의 대역폭을 도출할 수 있다. 각 데이터 흐름의 데이터 대역폭은 다음과 같다.

Decoded picture *Decoded picture*의 데이터 대역폭은 F_{width}가 프레임의 가로 길이, F_{height}가 프레임의 세로 길이, FPS가 초당 프레임 수(frames per second) 일 때 식 (1)을 이용하여 계산 할 수 있다.

$$D = (F_{width} \times F_{height}) \times 1.5 \times FPS \quad (1)$$

F_{width}와 F_{height}와 곱은 프레임 내 화소의 개수를 나타낸다. 전체 프레임의 크기를 바이트단위로 나타내기 위하여 각 화소마다 1.5를 곱한다. 한 화소가 1.5바이트인 이유는 다음과 같다. HEVC의 main profile은 샘플링 형식이 4:2:0이고 비트깊이(bit depth)가 8 비트이다. 이 경우, 한 화소가 8비트 luma 샘플 하나, 2비트 chroma 샘플 두 개로 구성된다. 따라서 한 화소는 총 12비트, 즉 1.5바이트이다. 마지막으로 초당 프레임 개수를 곱하면 *Decoded picture*의 데이터 대역폭이 도출된다.

HEVC bitstream *HEVC bitstream*의 데이터 대역폭은 식 (2)처럼 *Decoded picture*의 데이터 대역폭에 부호화 압축률 C만 곱하면 된다. 여기서 압축률 C는 부호화 결과인 bitstream의 용량을 원본영상의 용량으로 나눈 것으로 정의한다.

$$B = (F_{width} \times F_{height}) \times 1.5 \times C \times FPS \quad (2)$$

Reference data for MC *Reference data for MC*의 크기는 MC의 수행 횟수나 방법에 따라 유동적이다. 따라서 여기서는 최악의 경우의 대역폭을 구한다. 우리는 최악의 경우의 대역폭을 계산하기 위해 다음과 같은 가정을 하였다. 1) 각 PU의 예측화소는 intra prediction 또는 MC를 통해 예측되는데, 모든 PU의 화소값 예측이 MC를 통해 이루어진다고 가정한다. 2) 모든 PB는 쌍예측(bi-prediction)한다. 쌍예측 시에는 참조블럭이 두 개가 되므로 참조블럭이 한 장이 편예측(uni-prediction)에 비해 데이터 접근이 많다. 3) 모든 참조블럭은 2-D 보간을 이용하여 생성된다. HEVC의 MC는 예측 샘플의 정확도를 높이기 위하여 샘플 사이의 값을 보간을 통해 생성하여 참조하기도 한다. 이 샘플 사이의 값을 fractional 샘플이라고 한다. 보간은 주변 샘플들을 참고하여 수행된다. 보간 방법에는 1-D와 2-D가 있는데, 2-D가 1-D보다 참조하는 주변 샘플의 양이 많다. 3) 모든 PU의 크기는 4x4이다. 그 이유에 대해서는 뒤에서 설명한다. 4) 2-D interpolation 수행 시 가장 많은 샘플을 참조하는 필터를 이용한다. 2-D interpolation는 복수의 필터 중 하나를 이용한다. 우리는 모든 보간이 가장 많은 주변 샘플을 참조하는 필터를 적용한다고 가정하였다.

가장 많은 주변 샘플을 참조하는 필터를 적용한 2-D 보간의 참조 범위가 그림 9에 나타나 있다. 2-D 보간은 회색박스로 표현된 정수형 샘플 영역으로부터 가로, 세로로 half-pel 또는 quarter-pel단위로 이동한 영역에 빨간 사선 박스로 표시된 fractional 참조블럭을 생성하는 것이 목적이다. 이를 위해서 회색박스를 포함한 전체박스 영역 내의 정수형 샘플을 이용하여 보간을 수행한다. Luma 참조블럭을 생성하는 경우, 논문에서 가정한 최악의 경우에 필요한 주변 샘플은 그림 6과 같이 회색박스의 위쪽과 왼쪽으로 3 pixels내, 아래쪽과 오른쪽으로 4 pixels내에 있는 pixel이다. 이 범위는 PB의 크기에 관계없이 일정하다. 이 경우, PB의 크기가 작아질수록 참조하는 주변 pixel의 개수의 합이 늘어난다. 그런데 CTU가 작은 PU들로 분할될수록 PB가 작아진다. 이러한 이유 때문에 우리는 가장 작은 것으로 간주되는 4x4단위의 PU를 최악의 경우로 가정하였다. 실제로는 비대칭 분할을 하는 경우 PU의 크기가 4x4보다

작은 경우가 발생한다. 예를 들어 8x4크기의 유닛이 비대칭 분할을 통해 PU A와 B가 생성되는 경우, PU A의 크기가 4x4보다 작은 2x4가 되는 경우가 발생할 수 있다. 하지만 이런 경우 PU B의 크기는 PU A가 6x4로 PU A가 4x4보다 작아진 만큼 커진다. 따라서 2-D 보간 참조 샘플량의 관점에서는 4x4크기 PU 2개나 2x4, 6x4 크기 PU 2개나 동일하다.

*Reference data for MC*의 대역폭은 luma 참조 데이터 대역폭과 chroma 참조 데이터 대역폭의 합으로 구해진다. F_{width}가 프레임의 가로 길이, F_{height}가 프레임의 세로길이, *FPS*가 초당 프레임 수일 때, luma 참조 데이터에 해당하는 *Reference data for MC*의 대역폭은 식 (3)을 이용하여 계산할 수 있다.

$$R_l = \frac{(F_{width} \times F_{height})}{4 \times 4} \times (4+7)^2 \times 2 \times 1 \times FPS \tag{3}$$

F_{width}와 F_{height}와 곱은 프레임 내 화소의 개수이며 이를 luma PB의 샘플수로 나누어 주면 한 프레임 당 MC를 수행하는 PB의 개수가 된다. luma PB의 샘플수는 PU의 화소수인 4x4개와 동일하다. MC 수행 중 보간을 수행하기 위해 참조하는 정수샘플의 범위는 그림 9 (a)와 같이 가로는 PB크기+3+4이고 세로는 PB크기+3+4이다. 따라서 보간 작업이 참조하는 정수샘플의 개수는 (4+7)x(4+7)이다. 또한 PB는 쌍예측을 수행하기 때문에 한 MC를 두 번 수행하므로 2를 곱한다. 또한 한 luma 샘플의 크기인 1을 곱한다. 마지막으로 초당 프레임 수를 곱하면 대역폭이 계산된다.

Chroma 참조 데이터에 해당하는 *Reference data for MC*의 대역폭은 식 (4)을 이용하여 계산할 수 있다.

$$R_c = \frac{(\frac{F_{width}}{2} \times \frac{F_{height}}{2})}{2 \times 2} \times (4+3)^2 \times 2 \times 0.25 \times 2 \times FPS \tag{4}$$

식 (3)과 유사하나 다음과 같은 차이점이 있다. 먼저 chroma PB는 luma PB에 비해 가로, 세로 모두 길이가 절반이다. 따라서 F_{width}와 F_{height}를 2로 나누고, PB의 크기부분을 4x4에서 2x2로 바꾼다. 또한 MC 수행 중 보간을 수행하기 위해 참조하는 정수샘플의 범위는 그림 9 (b)와 같이 가로는 PB 크기+1+2이고 세로는 PB

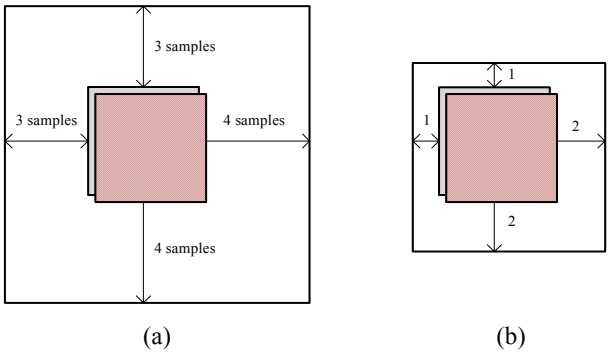

그림 9. 2-D 보간 수행 시 참조되는 샘플의 최대 범위 (a) luma 샘플 예측의 경우 (b) chroma 샘플 예측의 경우

Fig. 9. The maximum size of sample region referenced by 2-D interpolation. (a) In the case of luma samples prediction. (b) In the case of chroma samples prediction.

크기+1+2이다. 따라서 보간 작업이 참조하는 정수샘플의 개수를 (4+7)x(4+7)에서 (4+3)x(4+3)로 바꾼다. 또한 한 샘플의 크기를 chroma 샘플의 크기인 0.25바이트로 조정하고, chroma는 blue, red 두 종류가 있으므로 2를 곱한다.

식 (1), (2), (3)을 이용하여 하드웨어 HEVC 복호화기의 DRAM 접근량을 살펴본다. II장의 실험과 동일한 조건(F_{width} = 2560x1600, F_{height} = 1600, *FPS* = 30)을 가정한다. 압축률 C는 II장에서 사용한 bitstream중 QP가 37인 *Traffic* bitstream의 압축률인 1/575를 사용한다. 계산결과, 그림 8에 표시된 바와 같이 *HEVC bitstream*의 대역폭은 0.3MB/s, *Decoded picture*의 대역폭은 184.3MB/s, *Reference data for MC*의 대역폭은 2,234.8MB/s이다. 총 DRAM 접근량은 2,419MB/s이다.

3. 하드웨어 HEVC 복호화기와 소프트웨어 HEVC 복호화기의 메모리 접근량 비교

데이터 흐름량 분석을 통해 도출한 하드웨어 HEVC 복호화기의 DRAM 접근량 2,419MB/s은 III장에서 측정한 소프트웨어 HEVC 복호화기의 DRAM 접근량과 비교하면 상당히 적은 수치이다. 캐시 크기가 가장 큰 Core 2프로세서에서의 실험 결과에 따르면 HM 복호화기의 DRAM접근량은 6,940.7MB/s으로 하드웨어 복호화기의 DRAM접근량에 비해 2.9배 정도 많다. 데이터 흐름량 분석에 기반한 DRAM 접근량은 reference data의 양 측면에서 최악의 경우를 가정하였음을 고려하면

실제로는 차이가 더 클 것이다.

이러한 차이가 발생하는 원인은 캐시 자체의 비효율성과 소프트웨어 동작의 비효율성이 있다. 캐시 자체의 비효율성은 다음과 같은 이유로 발생한다. Cache line내에는 실제 HEVC 복호화기가 필요로 하는 데이터 외에도 불필요한 데이터가 존재한다. 그런데 DRAM과 cache간의 데이터 이동은 cache line단위로 발생하기 때문에 DRAM 접근량이 증가한다. 두 번째 원인은 소프트웨어 복호화기가 메모리 접근량 측면에서 비효율적으로 동작하는 것이다. 이러한 비효율성에는 소프트웨어에 불필요한 버퍼가 많이 존재하였거나, 데이터 크기보다 더 큰 저장공간의 할당 등이 있다. 실제로 HM의 경우, 8 bits sample 하나를 버퍼에 따라 short (2 bytes), int(일반적으로 4-8 bytes)등의 자료형을 이용하여 저장하고 있다. 소프트웨어 복호화기의 DRAM 접근량은 적절한 자료형 사용, 불필요한 버퍼 제거, 캐시 적중률 향상을 위한 코드 최적화, 더 큰 용량의 캐시 사용, scratch pad memory의 사용 등을 통해 감소시킬 수 있을 것이다.

V. 결 론

본 논문에서는 HEVC 복호화기의 메모리 접근량을 분석하였다. 먼저 시뮬레이션 및 실측을 통해 소프트웨어 HEVC 복호화기의 메모리 접근량을 측정하였다. 32KB의 데이터 캐시를 장착한 임베디드 프로세서에서는 평균 30.5GB/s의 DRAM접근이 발생하였고 데스크탑 프로세서에서는 6.9GB/s의 DRAM접근이 발생하였다. 또한 HEVC 복호화기의 데이터 흐름에 대한 이론적 분석을 수행한 결과 하드웨어로 HEVC 복호화기를 구현할 경우 이론적으로 최대 2.4GB/s의 데이터 접근이 발생하는 것으로 파악되었다. 소프트웨어 복호화기의 경우 자료구조 최적화, 캐시 메모리 크기 증가 등의 수정을 하면 데이터 접근량을 감소시킬 수 있을 것으로 파악된다.

REFERENCES

[1] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) Standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[2] H.-H. Jo, E.-K. Ryu, J. Nam , D.-G. Sim , D.-H. Kim , J.-H. Song, “Efficient parallel design for HEVC decoder,” *IEEK Summer Conference*, Jun. 2012, pp. 1940-1943.

[3] J. Ohm, G. J. Sullivan, H. Schwarz, Thiow Keng Tan, T. Wiegand, “Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC),” *IEEE Transactions on Circuits and Systems for Video Technology*, vol.22, no.12, pp.1669-1684, Dec. 2012.

[4] “AVC & HEVC: The Future of Encoding, Matthew Goldman,” TECHCON12.

[5] F. Bossen, “On software complexity decoding 720p content on a tablet,” JCTVC-J0128, ISO/IEC-JCT1/SC29/WG11, Jul. 2012.

[6] Y. J. Ahn, W. J. Han, and D. G. Sim, “Study of decoder complexity for HEVC and AVC standards based on tool-by-tool comparison,” *SPIE Applications of Digital Image Processing XXXV*, Proceedings of SPIE, vol. 8499, pp. 8499-32, San Diego, USA, August, 2012.

[7] Bossen, F.; Bross, B.; Suhring, K.; Flynn, D., “HEVC Complexity and Implementation Analysis,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol.22, no.12, pp.1685-1696, Dec. 2012.

[8] I.-K. Kim, J. Min, T. Lee, W.-J. Han, and J. Park, “Block Partitioning Structure in the HEVC Standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1697-1706, Dec. 2012.

[9] *The Magazine of the IEEK*, Vol. 38, No. 8, .Aug. 2011, pp. 40-45.

[10] Xtensa® Instruction Set Simulator (ISS) User’s Guide, 2011.

[11] F. Bossen, “Common HM test conditions and software reference configurations,” JCTVC-L1100, ISO/IEC-JCT1/SC29/WG11, Jan. 2013.

저 자 소 개

조 송 현(학생회원)
2007년 한양대학교 정보통신학부 졸업 (학사)
2009년 한양대학교 전자컴퓨터통신공학과 졸업 (석사)
2009년～현재 한양대학교 전자컴퓨터통신공학과 박사 과정
<주관심분야 : 멀티미디어 코덱, SoC 설계>

송 용 호(정회원)
1989년 서울대학교 컴퓨터공학과 졸업 (학사)
1991년 서울대학교 컴퓨터공학과 졸업 (석사)
2002년 University of Southern California, Electrical Engineering 졸업 (박사)
1991년～1996년 삼성전자 전임연구원
1996년～2002년 University of Southern California 연구조교
2002년～2003년 (주)조선 IT 연구소장
2003년～현재 한양대학교 융합전자공학부 교수
<주관심분야 : SoC 설계, 비휘발성 메모리, 멀티미디어 시스템, NoC 등>

김 영 남(학생회원)
2011년 강원대학교 전자공학과 졸업 (학사)
2011년～현재 한양대학교 전자컴퓨터통신공학 석사 과정
<주관심분야 : 멀티미디어 시스템, SoC 설계>