H.264 Encoder용 Direct Memory Access (DMA) 설계

정일섭* · 서기범**

*우송대학교

A design of Direct Memory Access For H.264 Encoder

Il-Sub Jung* · Kibum Suh**

*Electronic Dept. Graduate School, Woosong University

E-mail : isjung@wsu.ac.kr

요 약

본 논문에서는 Full 하드웨어 기반 베이스라인 프로파일 레벨 3규격 H.264 인코더 코덱에서 사용할 수 있는 Direct Memory Access (DMA)를 설계하였다. 설계된 모듈은 CMOS Image Sensor(CIS)로부터 영상을 입력 받아 메모리에 저장한 후 인코더 코덱 모듈의 동작에 맞춰 원영상과 참조영상을 각각 한 매크로블록씩 메모리에서 읽어 공급 또는 저장하며, 인코더는 한 매크로블록씩 처리하는데 660 cycle이 소요된다. 설계한 구조를 검증하기 위해 JM 9.4와 같은 reference Encoder C를 개발하였으며, Encoder C로부터 test vector를 추출하여 설계한 회로를 검증하였다.

ABSTRACT

The designed module save to memory after received Image from CMOS Image Sensor(CIS), and set a motion of Encoder module, read from memory per one macroblock each original image and reference image then supply or save. the time required 470 cycle when processed one macroblock.

For designed construct verification, I develop reference Encoder C like JM 9.4 and I proved this module with test vector which achieved from reference encoder C.

키워드

DMA, H.264, Encoder, SDRAM

Ⅰ. 서 론

영상을 연속적으로 처리하기 위해서는 큰 저장 장치가 필요하다. 예를 들어 HD영상의 4:2:0 포맷 1 프레임을 처리하기 위해서는 휘도 518,400워드 (1920x1080/4), 259,200워드가 생긴다. 통신간의 문제와 참조 영상까지 따지면 매우 큰 저장 공간이 필요한걸 알 수 있다. 본 논문에서는 SDRAM 1을 사용하여 베이스라인 프로파일 레벨 3규격의 H.264 인코더 코덱에 맞는 Direct Memory Access(DMA)를 설계하였다.[1][2]

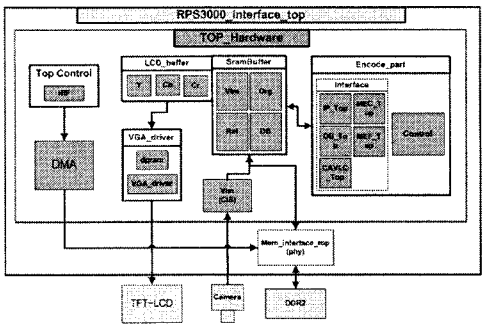


그림 1. DMA와 Encoder 전체블록 구조

DMA의 설계는 실시간으로 데이터 처리를 하는 인코더 코덱에서 매우 중요한 역할을 차지한다. 그 이유는 인코도 코덱이 짧은 시간에 많은 데이터 처리를 요구하기 때문이다. 블록 메모리 SRAM을 사용하면 동작 구조가 간단해서 데이터

1. 본 논문은 지식경제부 출연금으로 ETRI, SoC산업진흥센터에서 수행한 IT SoC 핵심설계인력양성사업의 연구결과입니다.

2. 본 논문의 내용을 발표할 때에는 반드시 ETRI, SoC산업진흥센터 IT SoC 핵심설계인력양성사업의 결과임을 밝혀야 합니다.

입출력도 빠르고 컨트롤이 쉽지만 저장 공간을 크게 만들 수 없다는 단점이 있어 사용할 수 없다. 하지만 SDRAM 1의 경우는 큰 저장 공간을 사용할 수 있지만 메모리 초기화, Cas 지연, Row 타이밍 등 제약 조건이 많다. 따라서 SDRAM 1을 사용하기 위해서는 인코더 코덱에 맞는 지연을 최소화 시킬 수 있는 메모리 구조와 컨트롤이 필요하다.

본 연구는 기존에 설계한 베이스라인 프로파일 레벨 3규격의 H.264코덱을 가지고 그림 1의 전체 설계구조와 같이 제한하는 SDRAM 메모리구조와 DMA컨트롤러를 적용하여 데이터 공급과 저장이 원활이 이루어지는 것과 인코더 처리가 확실한지 확인하는데 있다.

II. 인코더코덱과 DMA의 동작구조

제안된 메모리 구조를 설명하기에 앞서 기존에 설계된 베이스라인 프로파일 레벨 3규격 H.264 인코더 코덱을 간단히 설명하고자 한다.

설계된 인코더 코덱은 그림 2와 같이 5개의 (ME/F, Intra Prediction, CAVLC, Deblocking Filter) IP를 가지고 5단 파이프라인으로 영상을 처리한다. 한 IP의 동작 사이클은 660 사이클이며 5개의 파이프라인을 통과한 데이터가 인코더 코덱에서 한 매크로블록을 처리한 결과이다. 인코더 코덱에서는 한번에 5개의 IP가 동시에 수행되어 진다.

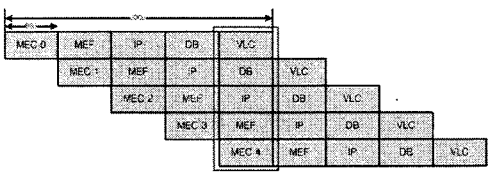


그림 2. 인코더 5단 파이프라인

DMA는 인코더 코덱의 5개 IP에 대한 데이터의 입출력을 책임지며 COMS 이미지센서로부터 데이터를 받아 메모리에 저장과 인코더 코덱에서 처리되어 저장된 영상을 읽어 LCD에 공급해주는 총 7번 메모리를 컨트롤한다.

그림 1을 보면 Top Control부분이 있는데 이곳에서 DMA와 인코더 코덱사이를 관장하여 인코더 코덱의 660사이클 동안 DMA가 7개의 IP에 데이터를 공급해주지 확인을 한다. 만약 660사이클 안에 DMA처리가 늦어졌을 경우 인코더 코덱은 WAIT 상태가 되어 DMA 처리가 완료되길 기다린다.

III. 메모리 구조

이번 장에서는 본 논문에서 말하고자하는 인코더 코덱에 최적화된 메모리 구조를 설명한다.

그림 3의 전체 메모리구조를 보면 원 영상 저장 부분은 하나를 사용하고 참조영상은 두 개를 저장하도록 구조를 만들었다. 원 영상의 경우에는 COMS 이미지센서의 동작 주파수가 24MHz에서 동작한다. 따라서 인코더 코덱이나 DMA와 상대적인 속도차가 심해 굳이 두 개를 사용할 필요가 없지만 참조영상의 경우에는 인코더 코덱의 ME/F에서 사용할 이전 처리 영상과 현재 영상을 처리해서 저장하는 곳이 따로따로 있어야하므로 두 개 영역을 사용하였다.

그림 4는 원 영상과 참조영상의 매크로블록 구조를 보여주고 있다. 그림 4.a를 보면 휘도 성분에 0~15, 색차성분의 Cb, Cr 각각 0~3의 숫자는 매크로블록 안의 작은 블록을 나타내고 밑의 b와 c의 숫자와 매칭 된다.

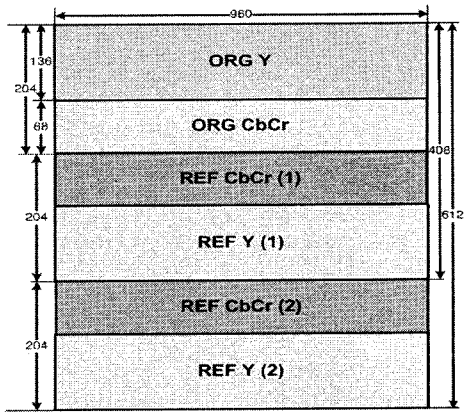


그림 3. 전체 메모리구조

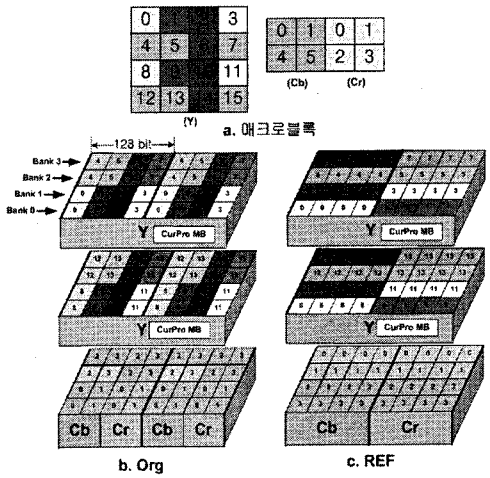


그림 4. Org, Ref MB 구조

그림 4.b는 원 영상을 보여주고 있다. 원 영상의 휘도성분은 메모리 뱅크 한 줄에 매크로블록 2줄을 저장하는 방법으로 4개의 뱅크에 매크로블록 8줄을 저장하여 한 매크로블록을 저장하는데 메모리 Row 주소를 2번 바꾼다. 색차성분은 휘도성분과 저장 방법은 같지만 2개의 성분을 번가라 가며 저장한다. 그림 4.c를 보면 알 수 있듯이 참조 영상은 원 영상과 저장 방법이 틀리다. 저장되는 크기는 원 영상과 같지만 구조는 메모리 뱅크 한 줄에 블록 전체를 저장하는 구조이다. 참조영상의 색차성분을 보면 뱅크3에서부터 0블록을 저장하도록 되어 있다.

원 영상의 경우에는 메모리로부터 읽어서 버퍼에 저장했다가 변환없이 인코더 코덱 IP에 공급시켜줄 수 있지만, 참조 영상의 경우는 메모리로부터 읽어서 버퍼에서 저장한 후 인코더 코덱 IP에 공급할 때 인코더 코덱이 처리 가능한 구조로 변환해서 공급해 줘야 하는 번거로움이 발생한다. 하지만 이런 구조를 가짐으로써 메모리 엑세스 딜레이를 최소화 시킬 수 있었다. 메모리 구조에 대해서는 다음 장에서 설명한다.

IV. 메모리 저장과 읽는 방법

이번 장에서는 메모리에 저장하는 방법과 저장된 데이터를 읽어가는 방법에 대해 설명한다.

메모리에 저장하는 경우는 두 가지와 읽어내는 경우 3가지가 있다.

원 영상을 저장하는 경우에는 COMS 이미지센서(CIS)로부터 라인으로 들어오는 영상 정보를 받아 그림 5와 같이 뱅크 하나에 두 라인을 저장하는데 128bit를 저장하고 128bit를 띄어서 다음 라인이 저장할 수 있는 영역을 남겨놓는다. 한 뱅크에 두 라인씩 저장하여 4개의 뱅크에 8라인을 저장한다. 색차 성분도 같은 방법으로 저장하며 Cb, Cr 순으로 저장을 한다. 읽어내는 경우에는 그림 4.b의 매크로블록의 휘도 성분을 읽어내는 방법은 한 뱅크에서 128bit를 두 번 읽어 들이고 4번의 뱅크를 바꾼 후 Row 주소를 1 증가하여 다음 블록을 읽어 매크로블록 크기만큼 읽어낸다. 색차성분도 똑 같이 읽어내며 다른 점은 Cb, Cr을 같이 읽어 낸다는 것이다..

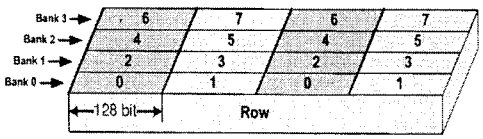
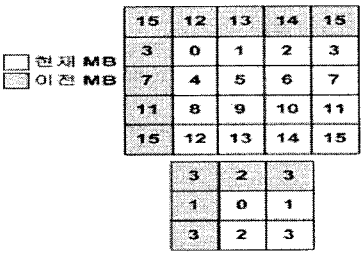
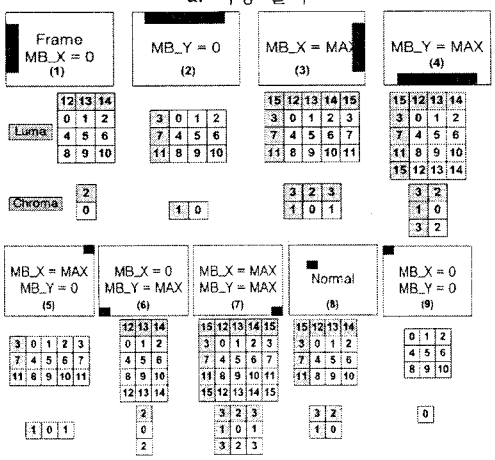


그림 5. 원 영상 저장 방법

참조 영상을 저장하는 경우는 인코더 코덱에서 Deblocking Filter(DB)을 끝낸 매크로블록을 저장한다. 저장되는 영상 정보는 그림 6.a와 같이 한 매크로블록만 저장하는 것이 아니라 이전 매크로블록도 저장해야 된다. 저장할 때도 경우의 수가 총 9가지가 있다. 그림 6.b의 1~9는 영상에서 매크로블록의 위치에 따라 저장되는 블록을 보여주고 있다. 최소로 저장되는 경우는 영상의 처음 시작 매크로블록일 경우 휘도 9개와 색차2개해서 11개 블록이 저장되고 최대일 경우는 영상의 마지막 매크로블록인데 휘도 25개, 색차 9*2개로 43개 블록을 저장한다.



a. 저장 블록



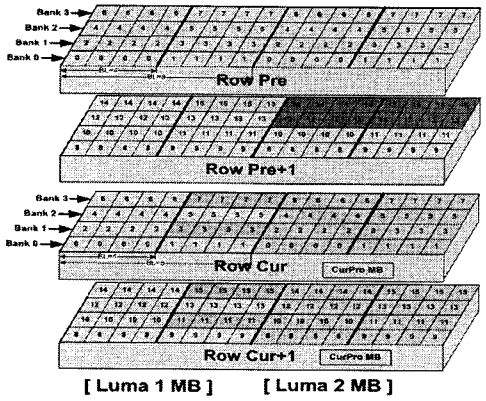
b. 저장 조건

그림 6. Deblocking Filter

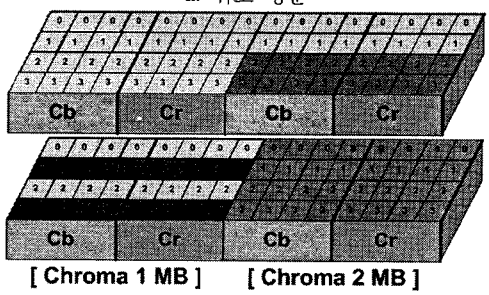
DB를 저장 시에는 매크로블록 단위가 아닌 블록 단위로 저장을 한다. 블록단위로 저장할 경우 원 영상 구조에서는 메모리의 Row 주소를 수시로 변경해야 해야 저장이 가능하다. 이렇게 되면 메모리에 데이터를 쓰는 시간보다 메모리 뱅크를 여는데 걸리는 시간이 더 걸리게 된다. 그래서 DB에서 저장할 때 그림 4. c처럼 저장을 하도록 변경하였다.

블록을 같은 Row 주소에 배치하여 DB의 9 조건에 의해 저장되는 블록 개수가 변한다 해도 Row 주소 변경이 원 영상처럼 많이 발생하지 않는다. 그리고 색차 성분을 저장할 때 휘도 성분과 다르게 반대로 저장하는 것은 휘도 성분의 이전 블록을 저장하는 부분(그림 7.a Row Pre+1영역의 12~15부분)을 저장할 때 뱅크 2~3을 사용한다. 이렇게 저장한 후 색차 성분을 저장할 때 다시 2~3

뱅크를 사용하여 뱅크를 열고 닫는데 지연이 발생한다. 이 부분을 없애기 위해 색차성분 저장을 휘도와 반대로 하도록 변경하였다.



a. 휘도 성분



b. 색차 신호

그림 7. 메모리에 저장되는 영역

V. 실험 결과

본 논문에서 제안한 메모리 구조로 RTL 설계를 하여 시뮬레이션 해본 결과 462사이클을 얻었다. 본 실험은 SDRAM1 166MHz을 기준으로 계산한 결과이다. 사이클 수는 메모리 종류와 속도에 따라 달라 질 수 있다. 그림 8을 보면 원 영상(VIM)을 저장하는데 105, 원 영상(Org) 읽는데 60, 참조영상(Ref) 읽는데 157, 참조영상을 저장하는데 134 사이클이 소요되는 것을 볼 수 있다.

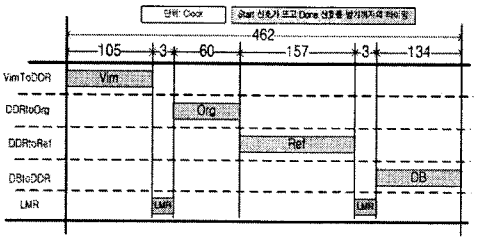


그림 8. SDRAM 최대 동작 사이클

VI. 결 론

본 연구는 베이스라인 프로파일 레벨 3규격에 맞는 최적화된 메모리 구조와 안정적으로 H.264 인코더 코덱과 통신하는데 있다. SDRAM의 뱅크 오픈을 줄여서 처음 데이터를 읽을 시에만 RAS 지연이 걸리도록 구조를 만들었다.

RTL 시뮬레이션은 Mentor graphics사의 modelsim을 이용하여 확인하였다. 시뮬레이션 결과 인코더 코덱과 통신이 안정적으로 이루어지고 있음을 확인하였고 Synopsys Desing Compiler를 통해 합성 결과 chartered 0.18공정 사용 시 인코더 코덱은 100MHz, DMA는 166MHz의 동작스피드를 가지며 173만 게이트크기의 결과를 얻었다.

참고문헌

[1] Joint Video Team, Draft ITU-T Recommendation and Fi-nal Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, May 2003.

[2] micron double data rate sdram, 512Mb_DDR_x4x8x16_D1.fm - 512Mb DDR: Rev. L; Core DDR Rev. A 4/07 EN.