H.264 Encoder용 Direct Memory Access (DMA) 제어기 설계

송인근*

A Design of Direct Memory Access (DMA) Controller For H.264 Encoder

In-keun Song*

요 약

본 논문에서는 Full 하드웨어 기반 베이스라인 프로파일 레벨 3 규격 H.264 인코더 코덱에서 사용할 수 있는 Direct Memory Access (DMA) 제어기를 설계하였다. 설계한 모듈은 CMOS Image Sensor(CIS)로부터 영상을 입력 받아 메모리에 저장한 후 인코더 코덱 모듈의 동작에 맞춰 원영상과 참조영상을 각각 한 매크로블록씩 메모리로부터 읽어서 공급하거나 저장하며, DMA 제어기의 한 매크로블록씩 처리하는데 478 cycle을 소요한다. 설계한 구조를 검증하기 위해 JM 9.4와 호환되는 Reference Encoder C를 개발하였으며, Encoder C로부터 Test Vector를 추출하여 설계한 회로를 검증하였다. 제안한 DMAC 제어기의 Cycle은 Xilinx MIG를 사용한 Cycle 보다 40%의 감소를 나타내었다.

ABSTRACT

In this paper, an attempt has been made to design the controller applicable for H.264 level3 encoder of baseline profile on full hardware basis. The designed controller module first stores the images supplied from CMOS Image Sensor(CIS) at main memory, and then reads or stores the image data in macroblock unit according to encoder operation. The timing cycle of the DMA controller required to process a macroblock is 478 cycles. In order to verify the our design, reference-C encoder, which is compatible to JM 9.4, is developed and the designed controller is verified by using the test vector generated from the reference C code. The number of cycle in the designed DMA controller is reduced by 40% compared with the cycle of using Xilinx MIG.

키워드

직접 메모리 접근 방식, H.264, 인코더, SDRAM

Key word

DMA, H.264, Encoder, SDRAM

* 우송대학교 철도전기 시스템학과

접수일자 : 2009. 09. 11
심사완료일자 : 2009. 11. 25

Ⅰ. 서 론

영상을 연속적으로 처리하기 위해서는 큰 저장장치가 필요하다. 예를 들어 HD영상의 4:2:0 포맷 1 프레임을 처리하기 위해서는 휘도 518,400워드(1920x1080/4), 색도 259,200워드가 필요하다. 통신간의 문제와 참조 영상까지 고려하면 매우 큰 저장 공간이 필요하다.

본 논문에서는 SDRAM 을 사용하여 베이스라인 프로파일 레벨 3 규격의 H.264 인코더 코덱에 맞는 Direct Memory Access(DMA) 제어기를 설계하였다[1,2].

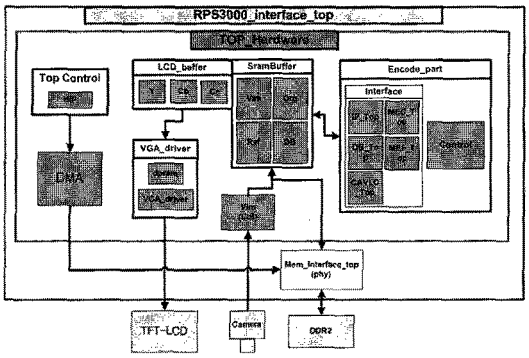


그림 1. DMA 제어기와 인코더 전체 구조
Fig. 1. The Architecture of DMA Controller and Encoder

DMA 제어기의 설계는 실시간으로 데이터 처리를 하는 인코더 코덱에서 매우 중요한 역할을 차지한다. 그 이유는 인코더 코덱이 짧은 시간에 많은 데이터 처리를 요구하기 때문이다. 블록 메모리 SRAM을 사용하면 동작 구조가 간단해서 데이터 입출력도 빠르고 제어가 쉽지만 저장 공간을 크게 만들 수 없다는 단점이 있어 사용할 수 없다. 반면에 SDRAM의 경우는 큰 저장 공간을 사용할 수 있지만 메모리 초기화, CAS(Column Address Strobe) 지연, Row 접근 타이밍 등 제약 조건이 많다. 반면에 SDRAM 을 사용하기 위해서는 인코더 코덱에 맞는 지연을 최소화 시킬 수 있는 메모리 구조와 제어가 필요하다.

본 연구에서는 기존에 설계한 베이스라인 프로파일 레벨 3 규격의 H.264 코덱을 가지고 그림 1의 전체 설계 구조와 같이 제안하는 SDRAM 메모리 구조와 DMA 제어기를 적용하여서 데이터 입출력이 원활이 이루어지는 가와 인코더 처리가 확실하게 수행되는 가를 확인하는데 목적이 있다.

Ⅱ. 인코더와 DMA 제어기 동작구조 및 메모리 특징

제안한 메모리 구조를 설명하기에 앞서, 기존에 설계한 베이스라인 프로파일 레벨 3 규격 H.264 인코더 코덱[3]을 간략이 설명한다.

설계한 인코더 코덱[4]은 그림 2와 같이 5단계의 정수형 움직임 추정 (MEC: Motion Estimation Coarse), 움직임 추정(MEF: Motion Estimation Fine), 화면내예측(Intra Prediction)[5], 내용기반 적응형 가변길이 코딩(CAVLC: Context-based Adaptive Variable Length Coding)) 모듈을 가지고 5단 파이프라인으로 영상을 처리한다. 한 모듈의 동작 사이클은 660 사이클이며 5개의 파이프라인을 통과한 데이터가 인코더 코덱에서 한 매크로블록을 처리한 결과이다. 인코더 코덱에서는 한번에 5개의 모듈을 동시에 수행한다.

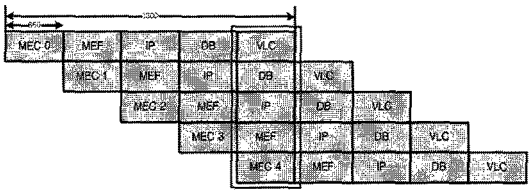


그림 2. 5단 파이프라인 인코더
Fig. 2. Encoder with 5 Stage Pipelines

DMA 제어기는 인코더 코덱의 CAVLC를 제외한 4개 모듈에 데이터의 입출력과 CMOS 이미지 센서로부터 데이터를 받아 메모리에 저장하여 총 5번 메모리를 제어한다.

그림 1의 Top Control에서는 DMA 제어기와 인코더 코덱 사이를 관장하여 인코더 코덱의 660 사이클 동안 DMA 제어기가 5개의 모듈에 대한 입출력을 원활히 수행하는가를 확인한다. 만약 660 사이클 안에 DMA 제어기 처리가 늦어졌을 경우, Top Control은 인코더를 WAIT 상태로 만들어 DMA 제어기 처리가 완료되기 전까지 기

다리도록 하였다.

메모리의 특징은 Active라는 명령을 이용하여, 사용하고자 하는 뱅크의 Row 주소를 열고, 일정 시간의 지연을 갖는다. 지연 시간이 경과해야만 Column 명령을 할 수 있고 Column 명령 또한 일정 시간 지연을 갖는다[2].

그림 3은 메모리에서 사용하는 명령어의 일부분 이다. 그림에서 위의 부분은 Active 명령의 사이클 지연이고, 중간과 밑의 그림은 읽기/저장 명령이다. 읽기는 저장 보다 더 긴 사이클 지연을 가지고 있음을 알 수 있다. 이와 같은 메모리 사용을 위해서는 사이클 지연을 고려한 메모리 구조를 만들어야한다. 메모리 구조를 어떻게 만드는 가에 따라 명령 횟수가 줄어들거나 늘어난다. (SDRAM의 명령어 지연은 동작 속도에 따라 달라진다.)

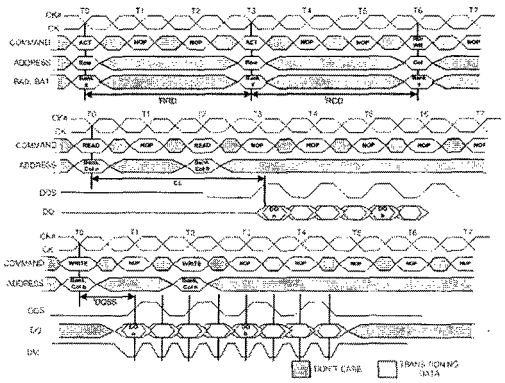


그림 3. 메모리 명령어
Fig. 3. Memory Instruction

III. 버퍼 구조

3장에서는 그림 1에서 기술한 Sram 버퍼에 대해서 설명한다. 그림 4는 인코더와 메모리 사이에 존재하는 Sram 버퍼의 구조와 데이터 흐름을 나타내었다. Sram 버퍼는 4개의 모듈로 구성 되어 있고, 듀얼 버퍼링이 가능하도록 여러 개의 Sram을 갖고 있다. DB 저장 버퍼는 인코더에서 Deblocking Filter를 끝낸 데이터를 메모리에 저장하기 위해 사용하며, VIM(Video Input Module) 저장 버퍼는 이미지 센서로부터 들어오는 데이터를 메모리에 저장하기 위해 사용하고, 원 영상을 저장하는 ORG(Original Source) 읽기 버퍼와 참조영상을 저장하는 REF(Reference) 읽기 버퍼는 인코더에 데이터를 넘겨주기 위해 사용하는 버퍼이다. 메모리 관점에서 저장에 사용하는 버퍼는 인코더 내부의 하나의 모듈과 관련하여 사용하고 있지만 읽기 버퍼는 인코더의 3개의 모듈과 (MEC, MEF, Intra Prediction) 관련하여 사용하므로 저장하는 버퍼보다 Sram를 많이 사용한다. 이와 같은 이유는 메모리의 읽기를 최소화 하기위해서이다. 인코더의 5단 파이프라인 구조(그림 2)에서 MEC, MEF, Intra Prediction 3개의 모듈에 공급하기 위해 매번 3번씩 메모리를 읽기 보다는 이전 MEC에서 사용했던 데이터는 다음에 처리할 MEF에서 사용하고, MEF에서 사용한 데이터는 다시 Intra Prediction에서 사용할 수 있도록 Sram을 여러 개 사용하였다.

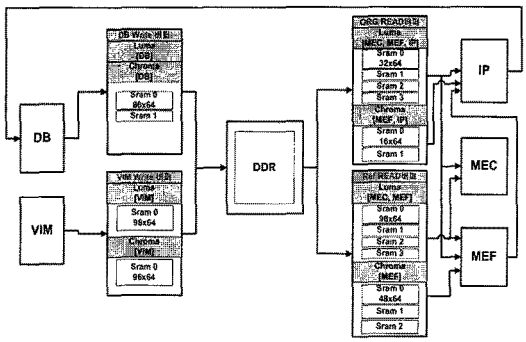


그림 4. Sram 버퍼 구조
Fig. 4. The Architecture of Sram Buffer

IV. 메모리 저장과 읽기 구조와 실험

4.1. 메모리 구조

4장에서는 본 논문에서 구현하고자 하는 인코더 코덱에서의 최적화된 메모리 구조를 설명한다.

그림 5는 고화질(HD; High Definition) 영상을 저장 할 수 있는 공간의 메모리 구조로서 하나의 원 영상과 두 개의 참조 영상을 저장한다. 그림 5는 메모리의 Column 주소가 960이며 HD 영상의 1920을 32 bits 단위로 나누면 (1 화소 당 8 bits) 480이다. Column 한 라인에 영상의 두 라인을 저장하는 구조이므로 두 배를 하면 960의 주소가 된다. Row 주소 크기는 4:2:0 포맷이기 때문에 휘도와 색도 성분의 주소크기가 다르다. 먼저, 휘도성분부터 계산하면 메모리가 4 뱅크이며 한 Column에 두 라인을 저장

하므로 1088을 8로 나눈 136의 주소가 된다. 색도성분은 휘도 성분을 2로 나눈 값인 68의 크기를 갖는다. 인코더에서 사용하는 메모리의 총 영역은 Column 960, Row는 휘도 3개, 색도 3개를 합한 612 주소이다.

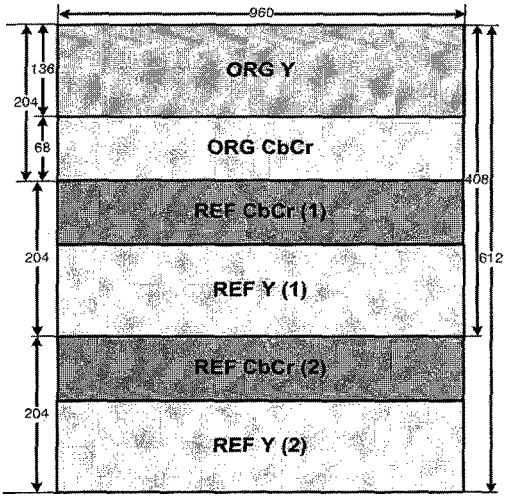


그림 5. 전체 메모리구조
Fig. 5. The Architecture of Whole Memory

그림 6은 원 영상과 참조영상의 매크로블록 구이다. 그림 6(a)는 휘도 성분에 0~15 값을 나타내며, 색차성분의 Cb, Cr에서의 0~3의 숫자는 매크로블록 안의 작은 블록을 나타내고 그림 6(b)와 6(c)의 숫자와 매칭 된다. 그림 6(b)는 원 영상으로 원 영상의 휘도성분은 메모리의 뱅크 한 라인에 매크로블록을 저장하는데 메모리 Row 주소를 2번 바꾼다. 색차성분은 휘도성분과 저장 방법은 같지만 2개의 성분을 번갈아 가면서 저장한다. 그림 6(c)에서의 참조 영상은 원 영상과 저장 방법이 다르다. 저장되는 크기는 원 영상과 같지만 구조는 메모리 뱅크 한 줄에 블록 전체를 저장하는 구조이다. 참조영상의 색차성분은 뱅크 3에서부터 0 블록을 저장한다. 원 영상의 경우에는 메모리로부터 읽어서 버퍼에 저장했다가 변환 없이 인코더 코덱에 공급 하지만, 참조 영상의 경우에는 메모리로부터 읽어서 버퍼에 저장한 후 공급할 때는 인코더가 처리 가능한 구조로 변환해서 공급하는 과정을 거쳐야 한다. 하지만 이런 구조로 인해서 메모리 액세스 지연은 최소화 할 수 있었다. 다음 절에서는 메모리에서 데이터 저장/읽기 방법에 대해서 설명한다.

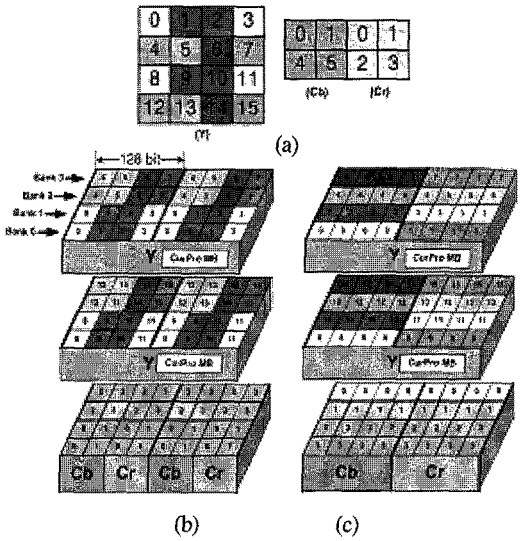


그림 6. 원 영상과 참조 영상의 매크로블록 구조
(a) 매크로블록 (b) 원 영상 (c) 참조 영상
Fig. 6. The Architecture of Macroblock in Original Image and Reference Image (a) Macroblock (b) Original Image (c) Reference Image

먼저, 원 영상의 저장과 읽기 방법부터 설명하고 이어서 참조 영상의 저장과 읽기 방법을 설명한다.

4.2. 원 영상 저장과 읽기 방법

원 영상의 저장과 읽기 방법에 대해 설명하면 다음과 같다.

4.2.1. 원 영상 저장 방법

원 영상은 CMOS 이미지센서(CIS)로부터 1화소(8 bits)단위씩 영상의 한 라인이 연속적으로 들어온다. 1화소를 4화소(32 bits) 단위로 만들어 4번 저장하여 128 bits를 메모리에 저장한다. 그림 7은 한 Row 주소의 4개 뱅크에 저장되는 영상의 라인이다. 그림에서 숫자 0, 1, 2, ..., 7은 영상의 라인수를 의미한다. 그림에서와 같이 한 Column에 두 개의 영상 라인이 저장되며, 같은 라인을 연속적으로 저장하지 않고 128 bits 단위로 끊어서 저장하였다. 영상의 짝수 라인은 Column의 0 bit부터 저장하고 홀수 라인은 128 bits 이후부터 저장한다. 따라서, 한 Row에 4개의 뱅크가 존재하므로 모두 8 라인을 저장할 수 있다. 색차 성분도 같은 방법으로 저장하며 Cb, Cr 순으로 저장을 한다.

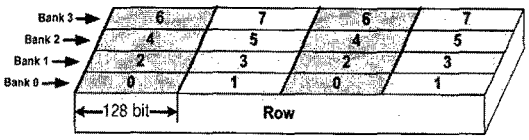


그림 7. 원 영상 저장
Fig. 7. Original Image Writing

4.2.2. 원 영상 읽기 방법

읽기 경우에는 인코더 코덱에서 요구하는 매크로블록 구조로 읽어야 한다. 그림 8(a)는 매크로블록의 구조이다. 한 매크로블록은 128 bits, 16 라인 이다. 한 Column에 128 bits씩 두 라인을 저장했기 때문에 256 bits를 읽어내면 매크로블록의 2 라인을 읽는 것과 같다. 그림 8(b)는 메모리에 저장되어있는 매크로블록의 구조를 보여주고 있다. 그림 8(a)에서 0~15의 숫자는 그림 8(b)의 숫자와 매칭 되며 같은 숫자가 4개가 존재하는 것은 4 라인을 의미한다. 메모리에서 매크로블록 읽기 방법은 한 뱅크에서 256 bits을 읽고 4번 뱅크를 바꿔가며 읽은 후 Row 주소를 바꿔서 같은 방법으로 읽는다. 색도성분도 같은 방법으로 읽으며 다른 점은 Cb, Cr을 같이 읽는 것이다.

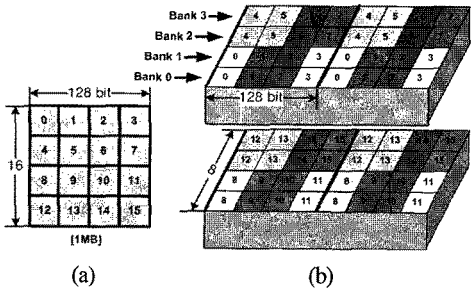


그림 8. 원 영상 읽기
(a) 매크로블록 구조 (b) 저장된 매크로 블록
Fig. 8. Original Image Read
(a) Macroblock Architecture (b) Macroblock Architecture (Writing)

4.3. 참조영상 저장과 읽기 방법

참조 영상을 저장하는 방법은 원영상보다 복잡하다. 인코더의 디블록킹 필터를 통해 나오는 데이터를 저장시, 디블록킹 필터는 블록 단위의 압축 방법을 사용하기 때문에, 발생하는 블록 간 경계점에서의 왜곡 현상을 제거하기 위한 모듈이라고 할수 있다. 블록과 블록의 경계 부분을 처리하기 때문에 여러 매크로블록이 섞여있다. 그림 9(a)는 위쪽으로 25개의 블록이 있으며 이것은 휘도 성분이다. 휘도 성분의 현재 매크로블록의 16개의 작은 블록과 위쪽으로 존재하는 이전 라인의 매크로블록인 12~15, 위에서 왼쪽으로 존재하는 15는 이전 라인 매크로블록의 직전 매크로블록을 나타내며, 현재 매크로블록의 왼쪽에 존재하는 것은 이전 매크로블록이다. 이와 같이 서로 다른 4개의 매크로블록이 존재한다. 색도성분 역시 블록의 개수는 다르지만 4개의 매크로블록이 존재한다. 이와 같이 여러 개의 매크로블록을 저장하면 Active 명령을 빈번히 수행해야 하기 때문에 실제 메모리에 저장하는 사이클보다 Active로 인해 지연이 더 발생한다.

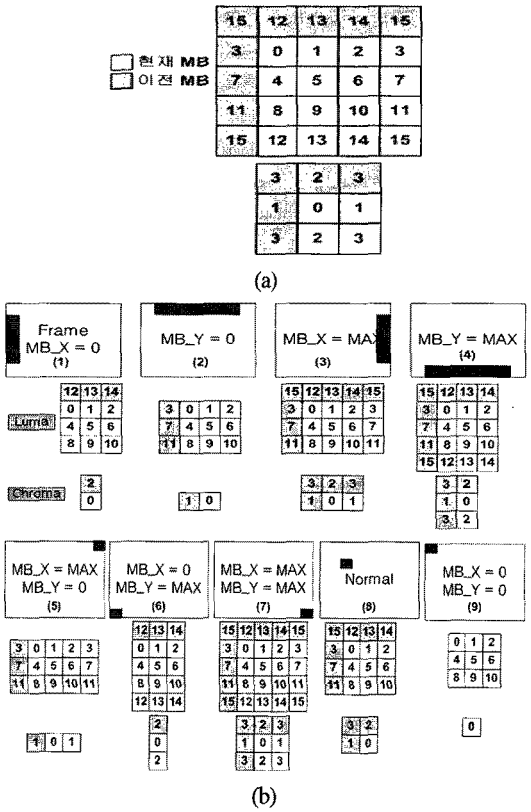


그림 9. 디블록킹 필터
(a) 저장 블록 (b) 저장 조건
Fig. 9. Deblocking Filter
(a) Block Writing (b) Writing Condition

그림 9(b)는 매크로블록이 영상의 위치에 따라 저장하는 조건을 그림으로 보여주고 있다. 영상의 각 모서리

지점 4개와 각 경계면 4개, 그리고 앞의 8가지 조건이 아닌 경우 1개로 모두 9가지 조건으로 저장되는 블록의 개수가 변한다. 최소 저장 블록은 11개이며 최대는 43개 블록이다. 9가지 조건에 맞춰 메모리에 저장하는 제어기를 만드는 것은 하드웨어 측면에서는 비효율적인 방법이므로 최대 블록 43개를 저장하는 하드웨어 제어기를 설계하였다.

4.3.1. 참조영상 저장 방법

앞에서 설명한 바와 같이 Active 명령 횟수를 줄이는 구조가 아니면 저장하는 사이클보다 뱅크를 여는 사이클이 더 소요 된다. 그림 10은 Active 명령 횟수를 줄이면서 메모리로부터 읽어가기 쉽도록 만든 메모리 구조이다. 원 영상과 저장 크기는 같지만 저장되는 매크로블록의 구조는 다르다. 참조영상의 경우는 블록단위로 저장을 하였다.

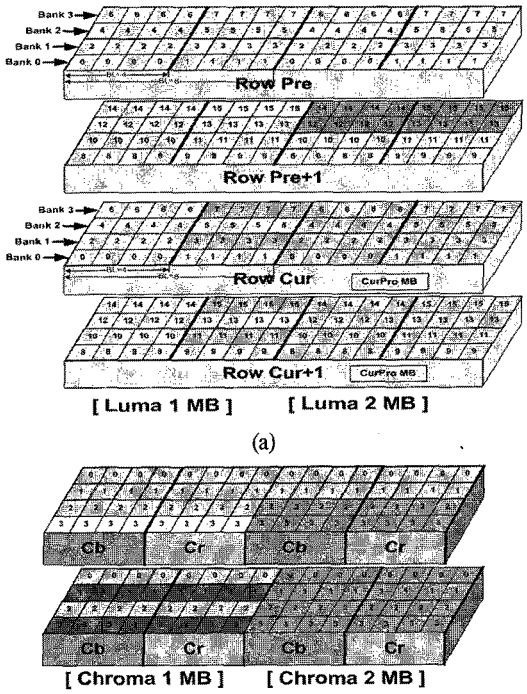


그림 10. 메모리에 저장되는 영역
(a) 휘도 성분 (b) 색차 신호
Fig. 10. Writing Area in Memory
(a) Brightness Component (b) Chrominance Signal

매크로블록은 작은 블록(4x4)으로 나누어진다. 하나의 블록이 32 bits씩 4라인(그림 10(a))으로 구성되기 때문에 일렬로 연결하면 128 bits의 크기를 갖는다. 이와 같이 한 블록을 길게 연결하여 같은 라인에 저장을 하도록 하였다. 그림 10은 같은 라인에 동일한 숫자를 4번씩 사용하고 있다. 그림 10에서는 색도 성분일 경우, 메모리에 저장되는 구조는 휘도 성분과 다르게 뒤쪽 뱅크부터 0 블록을 채우도록 하였다. 그 이유는 휘도 성분을 저장하고 나면 항상 2, 3번 뱅크에서 데이터 저장이 끝나게 되는데, 색도 성분을 휘도 성분처럼 저장을 하면 색도 성분도 항상 2, 3번 뱅크에서 저장을 마치게 된다. 따라서 같은 뱅크이지만 Row 주소가 틀리기 때문에 뱅크를 닫고 다시 열어야 저장이 가능하다. 이를 피하기 위해서 휘도 성분과 반대로 저장하는 구조로 하여 뱅크가 겹치지 않도록 하였다.

4.3.2. 참조영상 읽기 방법

원 영상은 라인을 128 bits씩 띄워서 저장했기 때문에 영상이 끊어져 있지 않아서 간단히 매크로블록을 읽어 인코더에 공급할 수 있다. 하지만 참조 영상의 경우에는 블록 단위로 저장을 했기 때문에 인코더에 공급하기 위해서만 변환을 하여야 한다. 그림 11에서 메모리로부터 데이터를 읽어서 4개 내부 Sram을 이용하여 변환을 시켜 인코더에 제공하고 있다.

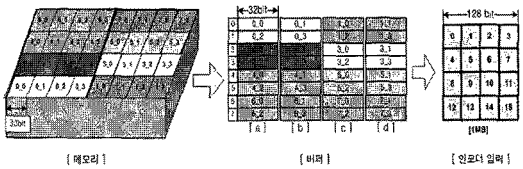
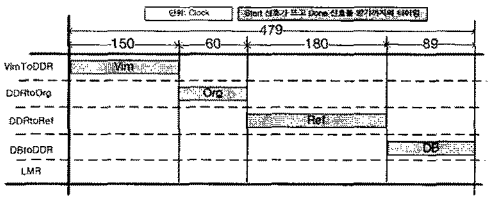


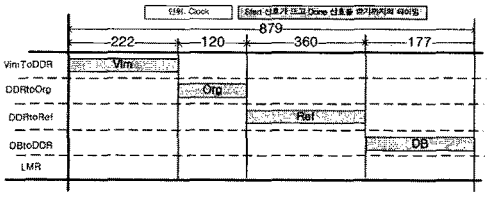
그림 11. 참조영상 읽기 방법
Fig. 11. Read Scheme from Reference Image

4.4. 메모리 구조 실험 결과

그림 12는 본 논문에서 제안한 메모리 구조로 RTL(Register Transfer Level) 설계를 하여 HD 영상을 기준으로 시뮬레이션 한 결과 사이클이다. 사이클 수는 메모리 종류와 속도에 따라 달라 질 수 있다. 그림 12(a)는 PHY(Physical PHY)만을 이용하여 메모리를 제어하는 사이클의 수이며, 그림 12(b)는 Xilinx MIG[6]를 이용하여 메모리를 제어하는 사이클 수이다.



(a)



(b)

그림 12. SDRAM 최대 동작 사이클
(a) PHY 만을 이용한 메모리 사이클
(b) Xilinx MIG를 이용한 메모리 사이클
Fig. 12. Maximum Operating Cycle of SDRAM
(a) Memory Cycle in case of only PHY
(b) Memory Cycle in case of Xilinx MIG

그림 12(a)와 12(b)의 사이클 수의 차이는 Row Activate, 읽기 명령으로 인한 tRRD, CL 지연으로 생긴 차이이다. 그림 12(a)의 경우 사이클 수는 그림 12(b)에 비해 상대적으로 작지만 SDRAM 166 MHz에서만을 고려하여 하드웨어 설계를 하고 시뮬레이션 검증을 하였고, 그림 12(b)는 Xilinx MIG(Memory Integration Generator)를 이용하여 범용으로 사용할 수 있도록 하드웨어를 설계하였다. 실험 결과의 검증은 JM과 호환되는 Reference C Code를 제작하여, 그 코드로부터 Vector 를 생성하여 그 결과를 비교하였다. ModelSim으로 시뮬레이션한 결과는 그림 13과 같다.

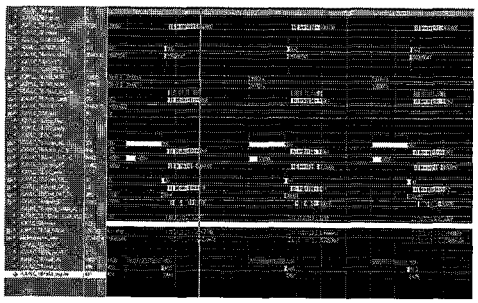


그림 13. Encoder 시뮬레이션 결과
Fig.13. The Simulation Results of Encoder

표 1. 사이클 계산 수식
Table 1. Computation Formula for The Number of Cycles.

종 류			수 식	비고
PHY	Vim	휘도	$(^tRRD)+(MB_x * Wclk * WnumY)$	99
		색도	$(^tRRD)+(MB_x * Wclk * WnumC)$	51
	Org	휘도	$(^tRRD)+(CL)+(Rclk * RnumY)$	38
		색도	$(^tRRD)+(CL)+(Rclk * RnumC)$	22
	DB		$(^tRRD)+(DBnum * Wclk)$	89
	Ref	휘도	$((^tRRD)+(CL)+(Rclk * RnumY)) * MBx$	114
		색도	$((^tRRD)+(CL)+(Rclk * RnumC)) * MBx$	66
MIG	Vim	휘도	$(^tRRD)+(MB_x * Wclk * WnumY)$ $+MB_x * WnumY(^tDQSS)$	147
		색도	$(^tRRD)+(MB_x * Wclk * WnumC)$ $+MB_x * WnumC(^tDQSS)$	75
	Org	휘도	$(RnumY * ^tRRD)+(CL * RnumY)$ $+(Rclk * RnumY)$	80
		색도	$(RnumC * ^tRRD)+(CL * RnumC)$ $+(Rclk * RnumC)$	40
	DB		$(DBact * ^tRRD)+(DBnum * Wclk)$ $+(DBnum * ^tDQSS)$	177
	Ref	휘도	$((RnumY * ^tRRD)+(CL * RnumY)$ $+(Rclk * RnumY)) * MBx$	240
		색도	$((RnumC * ^tRRD)+(CL * RnumC)$ $+(Rclk * RnumC)) * MBx$	120

표 2. 계산 수식 설명 (포맷 4:2:0)
Table 2. Explanation of Computation Formula (Format 4:2:0)

이 름	내 용	계	비고
^tRRD	SDRAM의 Row activate 사이클	3 clock	
^tDQSS	SDRAM의 저장 지연	1 clock	
CL	SDRAM의 읽기 지연	3 clock	
MB_x	매크로블록의 개수	3개	
$Wclk$	SDRAM에 저장하는 사이클 (burst length: 4)	2 clock	
$WnumY$	저장되는 휘도성분 블록 수	16개	
$WnumC$	저장되는 색도성분 블록 수	8개	
$Rclk$	SDRAM에서 읽어가는 사이클 (burst length: 8)	4 clock	
$RnumY$	읽어가는 휘도성분 블록 수	8개	
$RnumC$	읽어가는 색도성분 블록 수	4개	
$DBact$	참조영상 저장할 때 Row activate 횟수	16개	
$DBnum$	참조영상 최대 저장 블록 수 (휘도, 색도 포함)	43개	

그림 12에서 보여주는 사이클 수는 표 1의 수식을 이용해 SDRAM 166 MHz 기준으로 계산한 결과이다. 표 2는 표 1에서 사용한 문자를 설명한다. MIG를 사용하지 않고 직접 SDRAM을 제어하는 제어기를 설계하여 약 40%의 사이클을 줄일 수 있었다. 설계된 제어기는 MIG에 비하여 Row 접근시간을 최적화함으로서, Total Cycle을 줄일 수 있었다.

V. 결 론

본 논문의 메모리 구조는 베이스라인 프로파일 레벨 3 규격 H.264 인코더 코덱에 적합한 최적화된 구조이고 메모리 구조에 적합한 DMA 제어기는 안정적으로 H.264 인코더 코덱과 통신하였다. 본 논문에서는 SDRAM의 Active 명령을 줄이기 위해 한 Column에 두 라인을 저장하여 Active 명령을 반으로 줄였다. 표 3은 합성 결과이다.

RTL시뮬레이션은 Mentor Graphics사의 ModelSim을 이용하여 확인하였으며, 시뮬레이션 결과 인코더 코덱과 통신이 안정적으로 이루어지고 있음을 확인하였고, Synopsys Design Compiler를 통해 합성 결과 Chartered 0.18 공정 사용시 Encoder 코덱은 108MHz 이고, DMA 제어기는 166MHz의 동작속도를 가지며 173만 게이트 크기의 결과를 얻었다. 하지만, 향후 UDTV를 지원하기위해서는 DMA 제어기의 성능향상이 필요하다.

표 3. 합성 결과
Table 3. Synthetic Results

합성툴	동작속도	Gate
Synopsys Design Compiler Chartered 0.18공정	인코더 : 108MHz	173만
	DMA 제어기: 166MHz	

참고문헌

[1] Joint Video Team, Draft ITU-T Recommendation and Final 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.

[3] Yu-Wen Huang, Bing-Yu Hsieh, Tung-Chien Chen, and Liang-Gee Chen, "Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL.15, NO.3, MARCH 2005

[4] 김종철, 서기범(2008), "H.264 Encoder Hardware Chip 설계", 한국해양정보통신학회 추계종합학술대회, Oct. 2008

[5] An Efficient Hardware Architecture of Intra Prediction and TQ/IQIT Module for H.264 Encoder Kibum Suh, Seongmo Park, and Hanjin Cho, ETRI Journal, vol.27, no.5, Oct. 2005, pp.511-524.

[6] Xilinx Memory Interface Generator (MIG) User Guide

저자소개

송인근(In-keun Song)

1978년 고려대학교 전자공학과 (공학사)
1983년 고려대학교 전자공학과 (공학석사)
1999년 명지대학교 전자공학과(공학박사)
1984년~1995년 한국전자통신연구원 선임연구원
1995년~현재 우송대학교 철도전기시스템학과 교수
※관심분야:영상처리