고속 움직임 예측기의 FPGA 설계

*임정훈, *서영호, **최현준, *김동욱
*광운대학교, **안양대학교
jh_lim@kw.ac.kr

FPGA Design of High-Speed Motion Estimator

*Jeong-Hun Lim, *Young-Ho Seo, **Hyun-Jun Choi, *Dong-Wook Kim
*Kwangwoon University, **Anyang University

요약

본 논문은 H.264/AVC 디코더의 하드웨어 구현 시 가장 많은 시간을 소비하는 부분이 움직임 추정기를 하드웨어로 구현하였다. 움직임 추정을 함에 있어서 외부메모리 Access 량을 줄이고, SAD연산을 수행할 때 Clock의 손실 없이 계산을 하는 움직임 예측기를 제안한다. 제안한 구조는 재탐색 구간에서 이전 탐색 범위와 공통부분을 이루는 부분을 레지스터에 따로 저장해 두었다가, 재탐색시에 이전 Data를 사용하는 방법을 이용하였다. 움직임 추정을 수행할 때의 SAD (Sum of absolute differences)연산 부분과 Adder-tree를 묶은 PU Array와 SAD 누적기, 선택기를 Pipelining을 통하여 Clock의 손실 없이 연속적으로 계산하는 움직임 예측기를 설계하였다. 구현한 하드웨어는 최대 446.43MHz의 주파수에서 동작할 수 있었고, 탐색영역 64x64, 참조 프레임 3, 그리고 영상크기 1920×1080 기준으로 구현한 결과 50 프레임을 처리할 수 있는 성능을 보였다.

1. 서론

현대인들은 보다 나은 영상을 보려 하고 있다. 더욱이 최근 들어서 3DTV, UHDTV등의 고화질 영상들을 실시간으로 볼 수 있게 하기 위해서는 Motion Estimation의 연산을 보다 빠르게 처리 할 수 있도록 하는 연산기가 필요하다. H.264의 움직임 추정은 부호화기의 전체의 약 95% 이상을 움직임을 찾는데 연산과 메모리 접근에 사용하고 있다. 연산시간과, 메모리 접근을 줄이기 위하여 고속의 병렬 연산 구조에 관한 연구들과, 병렬 연산 구조의 메모리대역폭을 줄이기 위한 중간버퍼 관리, 검색윈도우관리, 움직임추정을 위해 데이터 재사용을 이용한 하드웨어구조, 다중 참조프레임의 정보를 재사용하는 방법 등이 제안되고 있다. [1][2]

기존의 다중참조프레임을 이용한 움직임 추정 방법들은 참조프레임의 개수만큼의 내부 메모리가 필요로 하게 되어 하드웨어로 구현하기에는 비효율적이다. 특히 외부 메모리의 접근은 크게 3가지의 요소에 큰 영향을 받게 되는데, 참조프레임, 탐색범위의 중심점, 그리고 탐색범위의 크기가 있다. 이러한 3가지의 요소들을 하드웨어로 구현하기에는 하드웨어의 복잡도가 높아져 구현하기에 어려움이 많이 있다. [3]

움직임 추정이란 동영상 비디오에서 각 객체나 단위블록(Macroblock)이 시간상 앞뒤 프레임에서 어느 위치로 움직였는지를 추정하는 것을 말하는 것으로 MPEG, H.264/AVC 등의 동영상 압축과 프레임 보간 등의 비디오 영상처리 기술에서 광범위 하게 이용되고 있다. H.264/AVC 압축표준에서는 매크로 블록 단위로 현재 압축하고자 하는 매크로 블록과, 앞뒤 여러 장의 프레임의 주변 매크로 블록과의 각 픽셀간의 명도차이를 초소로 하는 블록을 검색하여 움직임 벡터(Motion Vector)를 결정하게 된다. [4][5]

움직임 벡터 결정 후에 움직임 추정된 매크로 블록과 압축하고자 하는 매크로 블록간의 차와 움직임 벡터만을 압축하게 된다. 여기서 벡터 값을 추출하여 데이터를 전송 하는 과정에서 SAD 값 연산 부분이 가장 많은 계산을 수행하는 부분이다.

2. H.264의 인코딩

H.264/AVC 예측 부호화 기법중 하나인 화면 간 예측(Inter prediction)은 움직임 추정 및 보상을 통하여 입력 영상의 시간적 중복성을 제거하여 부호화 효율을 높이기 위한 기법이다. 움직임 추정은 화면 간 예측에 있어서 예측 부호화 후의 잔류 신호를 최소화하여 부호화 효율을 높이기 위해 반드시 필요한 부분이다. 움직임 추정 및 보상 과정에서 발생되는 움직임 정보는 각 화면 간 예측 모드별로 움직임 벡터(Motion Vector)와 참조 프레임(Reference Frame) 인덱스 정보, 그리고 해당 모드 번호로 구성된다. H.264/AVC 에서는 블록의 크기에 따라 Inter 16x16, Inter 16x8, Inter 8x8모드, 그리고 Inter 8x8 모드의 경우 8x8, 8x4, 4x8, 4x4 모드의 서브 매크로 블록(Sub-macroblock)으로 구성되어 있다. [6]

움직임 추정 (Motion Estimation)은 단위블록(Macroblock) 또는 Block 와 가장 유사한 것들을 이전 프레임에서 찾아 현재 프레임의 값을 추정하는 과정이다. Motion Estimation의 목적은 되도록 많은 단위 블록 또는 블록을 이전 프레임에서 참조함으로써 중복성을 없애서 압축률을 증가시키는데 그 목적을 둔다. 특히 H.264는 지난 표준 비디오 압축방식 과는 달리 하나 이상의 참조영상을 움직임 보상에 이용 한다. 다양한 가변 블록단위의 움직임 추정을 통한 복잡하고 정밀한 움직임 영역을 표현할 수 있게 되었다. 그러나 이러한 여러 개의 참조 영상과 가변블록은 부호화기의 움직임 예측시 최적의 움직임 벡터를 찾아내기 위한 반복적인 탐색을 요구하기 때문에 더 많은 연산량을 필요로 한다. [7]

3. 제안한 움직임예측 하드웨어

3.1 제안한 H/W의 특징

제안한 하드웨어의 구조 그림 1은 움직임 추정을 위한 SAD 값을 출력하는 방법이 병렬화 되어 있는데, PE(Processing Element : 4x4 SAD Module)라는 단위 연산기로 구성된다. 16개의 PE와 Adder-tree를 하나의 PU(Processing Unit) 로 묶어서 16x16 블록을 계산하고, 16개의 PU를 하나의 PUA(PU Array)로 구성한다. SAD 계산, 누적기, 그리고 선택기를 파이프라인화하여 PU0부터 PU15까지 연산을 진행하고, PU15의 연산이 끝남과 동시에 다음 프레임의 PU0 계산을 연속적으로 계산하여 지연시간 없이 탐색영역내의 SAD연산을 수행할 수 있는 구조로 되어 있다. 외부 메모리로부터 읽어 들이는 데이터는 현재 프레임 데이터(Current Frame Data)과 참조 프레임 데이터의 다음 탐색 범위를 계산 할 때에 이전의 탐색범위와 중복되는 부분을 레지스터에 저장해두었다가 새로운 탐색영역을 계산할 때 이전의 데이터를 재사용하는 방법의 하드웨어 구조를 제안한다.

3.2. 전체적인 하드웨어 동작

제안한 모듈의 구성요소는 외부 메모리로부터 현재프레임 데이터와, 참조프레임 데이터를 저장하는 2개의 버퍼로 구성되어져 있으며, PU Array 는 16개의 Processing Unit 으로 이루어져 있으며, 여기서 가변 블록 크기의 7가지 크기에 대하여 41가지 sub_block을 만들고, 블록들로부터 움직임 벡터를 선택하여 출력하는 구조로 이루어져 있다. 탐색범위는 64x64 에 16x16 단위블록 매칭방법으로 SAD 값을 출력한다. 현재와 참조프레임의 데이터를 각각의 PU(Processing Unit : 16개의 PE로 구성) 으로 입력되어 출력은 SAD 값으로 출력되어 SAD Processor를 거쳐 움직임 벡터를 찾아 가는 구조이다. MVP(Motion Vector Predictor)은 움직임의 이전의 중심벡터 값과 MVP의 차이 값이 작으면 이전의 중심에 고정이 되고, 두 값의 차이가 크면 MVP은 새로운 탐색 중심점을 찾아 가는 방법을 사용하였다.[8]

PU 계산 이동시 ME 동작 방식은 PU의 계산이 진행 될 때 단위블록의 동작 방식 그림2(a)에서 16x16 의 입력 데이터를 각각의 16개의 PE에 순차적으로 16x1의 데이터를 총 16 Cycle 안에 처리한다. 그림 2(b) PU의 동작순서는 16개의 PE의 계산을 (0,0)을 기준으로 (0,15) 까지 순차적으로 이동하면서 현재와, 참조프레임의 값을 계산 한다.

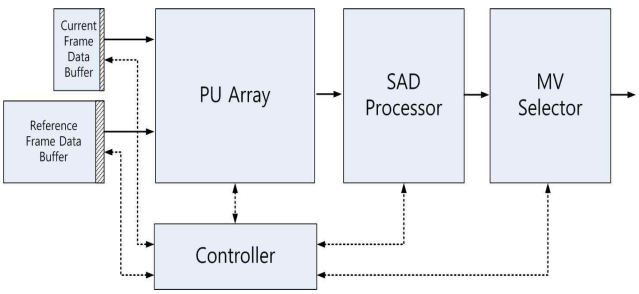

그림 1. 제안한 하드웨어의 구조

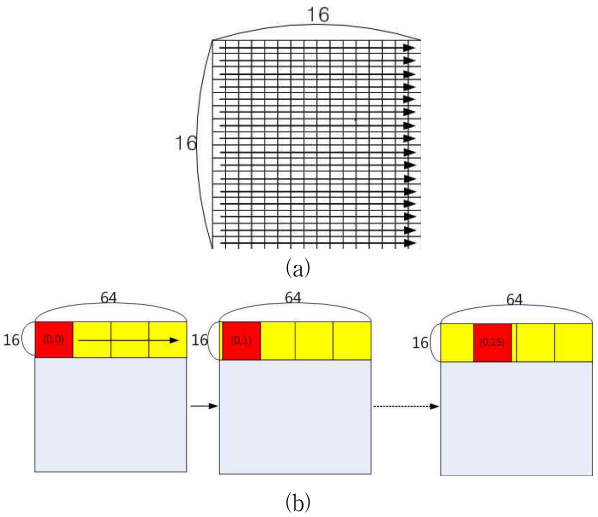

그림 2. Motion Estimation 의 동작 순서 (a) PE의 동작순서 (b) PU의 동작순서

3.3 제안한 하드웨어의 세부구조

가. Processing Unit(PU) Array 의 구조

그림 3은 하나의 PU는 총 16개의 PE(Processing Element)이 병렬연결로 연결되어져 있으며, 16개의 PU는 하나의 PU Array를 구성한다. 참조 단위블록의 데이터는 데이터의 흐름에 따라 16개의 PE 안에서 움직여지지만 현재 단위블록 데이터는 한번 입력이 되면 고정되어, 이동되거나 입력되어지는 참조 단위블록 데이터와 계산되어 SAD 값을 출력하게 된다.

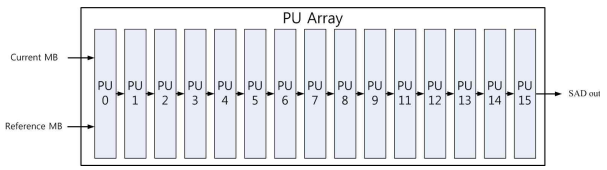

그림 3. PUA(Processing Unit Array)의 구조

나. Processing Unit 의 세부구조

그림 4는 PU 구조 이다. 하나의 PU는 16개의 PE와, Adder Tree로 이루어져 있다. 이러한 구조는 PE에서 SAD 값들을 7가지의 가변 블록의 값을 출력하게 하고, 41가지의 SAD 값들을 선택적으로 출력하는 구조로 이루어져 있다. PU 구조는 16개의 PE로부터 계산되어진 값들은 R1부터 R3 의 레지스터에 각각 저장되고, 7가지의 가변블록들의 값들을 출력하게 된다. R1에서 PE에서 계산된 초기 값(16x1)들이 계산되고, R2에서 PE0~PE3 까지의 SAD 연산들의 값을 이용하여 4x4 SAD, 4x8 SAD 값들을 출력하고 R3 에서는 8x4, 8x8, 8x16 SAD 값들이 조합되어 출력이 된다. R4 에서는 R3로부터 입력받은 SAD 데이터들을 이용하여 16x8, 16x16 SAD 값들을 출력한다.

다. Processing Element 의 세부구조

PE(Processing Element)의 구조는 16개의 PE는 그림 2(a)에서 입력받는 방법으로 데이터를 입력받아, 16개의 PE가 모여서 16x16 블록의 데이터를 처리하는 부분이며, 여기서 처리된 값들은 Adder-tree를 통하여 가변블록의 7가지 크기에 대하여 41가지 sub-block의 SAD

값을 출력하는 부분 그림 5 이다. 16개의 PE가 16x16의 256개의 픽셀 정보를 입력받아 PE의 SAD과정을 거쳐 그림 4의 레지스터 R1에서부터 R4 까지 각각의 계산된 값들이 저장되어지며, 각각의 레지스터에서는 7가지의 매크로 블록의 SAD를 출력하여 다음 과정으로 보내어진다.

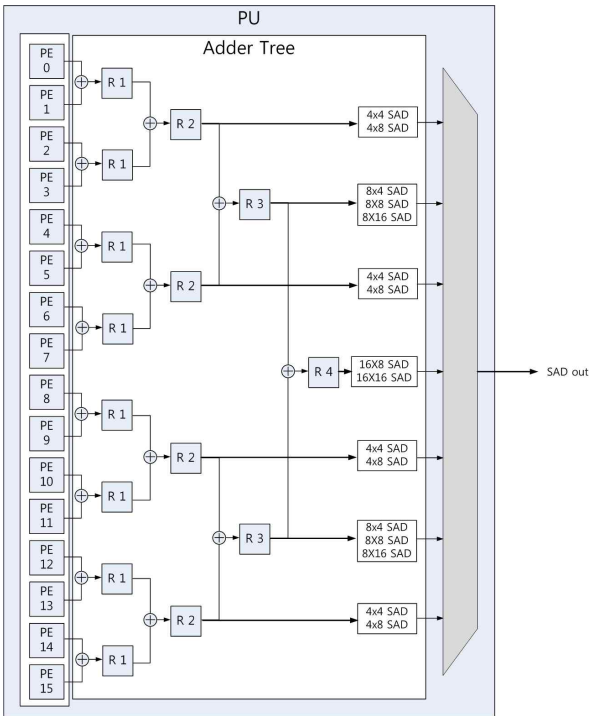

그림 4. Processing Unit 의 세부구조

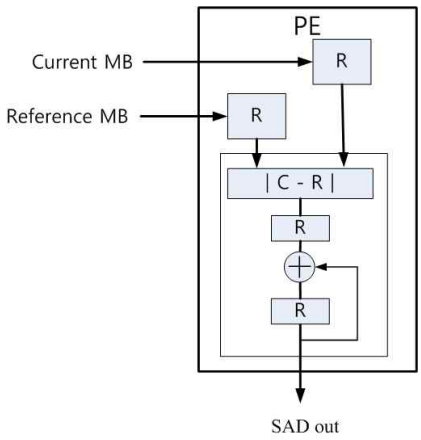

그림 5. Processing Element 의 세부구조

Processing Element 는 현재 프레임의 단위블록 과 참조 단위블록의 SAD을 출력하는 구조로 되어 있다. 각각의 PE는 현재 와 참조 매크로블록의 픽셀 데이터를 저장하는 2개의 레지스터로 구성되어있고, 이 두 값이 계산기 Unit 에 입력되어 각각의 픽셀에 대한 SAD 값을 구한다. 여기서 현재 단위블록에 대한 데이터 값은 변하지 않고, 참조부분의 데이터 값이 프레임의 개수에 따라 입력 받아 계산되어지게 된다. SAD module 은 현재, 참조 단위블록의 데이터 값을 가지고 SAD를 계산한다. 계산을 통해 구해지는 4x4 블록의 SAD를 이용하여 41 Motion Vector를 구할 수 있다.

위에서 설명한 각각의 요소들을 이용하여 PU 의 계산을 순차적으로 배열하여 매 사이클 마다 Clock의 손실을 방지하기 위해 그림 7의 PU를 Pipelining 하는 구조를 설계하였다. PU에서 SAD 값을 계산하면서 누적기를 통하여 SAD 값들을 누적하면서 출력되어지는 SAD 값들의 7가지의 가변블록 모드별 블록의 값을 선택적으로 출력해주는 SAD 선택기로 이루어져 있다. 이러한 순서로 Pipelining 이 이루어져 PU0에서 부터 PU15까지 하나의 PU Array의 연산이 진행되고, 다음 PU Array의 연산이 연속적으로 이루어지는 구조로 되어 Clock의 손실 없이 계산하는 구조이다.

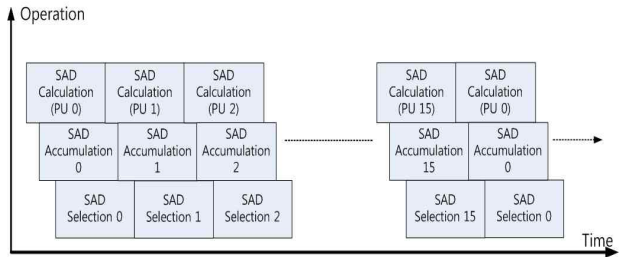

그림 7. PU의 Pipelining 구조

3.4 On-chip 메모리

그림 8(a)는 현재 단위블록에 적용한 데이터 버퍼링이다. 참조 프레임(RF) A를 사용할 때는 참조프레임(RF) B에 다음 데이터를 입력시킨다. RF-B를 사용 할 때에는 RF-A에 다른 데이터를 입력시킨다. 이와 같은 방법으로 인하여 하드웨어의 량은 32x16으로 반이지만 외부 메모리로부터 접근하는 횟수는 증가 한다. 그림 8(b) 참조 단위블록에 적용한 데이터 버퍼링이다. 처음 64x16 (0,0)~(0.63) 탐색 범위에서 움직임 추정을 하고 나서(1,0)~(1,63) 범위로 넘어갈 때, 이전 탐색 구간과 새로운 탐색범위는 서로 공통부분이 생긴다. 공통 데이터 64x15 부분이 서로 공통으로 사용되는 데이터 인데 이 부분의 값을 레지스터에 저장하여 재사용되는 부분을 그대로 두고 새롭게 탐색한 부분만을 외부 메모리로 부터 불러온다. 이렇게 현재와 참조 단위블록에 서로 다른 데이터 버퍼링을 이용하여 움직임 추정을 할 때 현대 단위블록의 값들은 고정되어 있어 그림8 (a)와 같은 방법을 사용하고, 참조 프레임에서는 다중 참조 프레임을 이용하여 움직임 추정을 하기에 그림8 (a)의 방법으로는 공통부분을 다시 읽어야 하기에 처리 속도, 데이터 사용 부분에서 손실이 발생함으로 그림 8(b)와 같은 방법을 이용하여 데이터 재사용방법을 이용하여 처리 속도와, 내부 메모리에서의 사용량을 줄일 수 있다.

4. 구현 결과

본 논문에서는 H.264 움직임추정의 외부메모리 접근 감소와, 참조 프레임 데이터를 재사용, PU Array의 Pipelining을 통하여 연속적인 계산하는 구조를 제안하였다.

움직임예측기는 Verilog 행위수준으로 설계하였으며, Altera의 Quartus II 환경을 사용하였다. 움직임예측 하드웨어의 Logic Utilization은 표 1, 움직임예측처리부의 RTL 합성도는 그림 9에 나타내었고 움직임 예측기의 동작 주파수는 446.43MHz 로 동작하였다. 타겟 플랫폼은 Altera의 StatixⅢ EP3SE80F1152C2칩이었다. 탐색영역

은 64x64에 16x16 단위블록으로 참조 프레임 3개 , 1920x1080p을 50 Frame/s를 처리하는 결과를 보였다.

5. 결론

본 연구에서는 H.264 움직임추정의 외부메모리 접근과, 참조프레임의 데이터를 재사용 , SAD 연산 부분을 Pipelining을 하여 Clock의 손실 없이 계산하는 하도록 하였다. 설계한 하드웨어는 면적의 효율성을 높여 FPGA에 성공적으로 사상되었다. 또한 본 논문에서는 설계 사상 목표를 정하지 않고 움직임추정 하드웨어를 범용적으로 설계하였다. 설계한 결과를 Alter FPGA인 StatixⅢ EP3SE80F1152C2에 사상한 결과 446.43MHz 주파수에서 안정적으로 동작하여 1080p 영상을 최대 50 Frame/s을 처리 할 수 있었으며, FPGA의 4%의 Combinational ALUTs와 3%의 Memory ALUTs만을 사용하여 낮은 하드웨어 사용률을 보였다. 이 설계를 ASIC으로 구현할 경우 더 높은 동작속도를 보일 것으로 예상되어 향후 본 논문의 결과는 고속 움직임 추정기가 필요한 많은 응용분야에서 효과적으로 사용될 것으로 기대된다.

감사의 글

본 연구는 한국산업기술평가관리원(KEIT)의 IT산업원천기술개발사업의 일환으로 수행하였음. [KI002058, 대화형 디지털 홀로그램 통합서비스 시스템의 구현을 위한 신호처리 요소 기술 및 SoC 개발]

참고문헌

[1] J.-C. Tuan, T.-S. Chang, and C.-W. Jen, “"On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," IEEE Trans. Circuits Syst. Video Technol., vol. 12, no. 1, pp. 61 - 72, Jan. 2002.

[2] Y. Su and M.-T. Sun, “"Fast multiple reference frame motion estimation for H.264/AVC,"" IEEE Trans. Circuits Syst. Video Technol., vol. 16, no. 3, pp. 447 - -452, Mar. 2006.

[3] Z. Liu, L. Li, Y. Song, T. Ikenaga, and S. Goto, “"VLSI oriented fast multiple reference frame motion estimation algorithm for H.264/AVC,"" in Proc. IEEE Int. Conf. Multimedia Expo, Beijing, China, pp. 1902 - 1905, Jul. 2007.

[4] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard", IEEE Trans. Circuits and Systems for Video Technology, vol.13, no.7, pp. 560-576, Jul. 2003.

[5] “Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC," in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050, 2003.

[6] 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.

[7] J. Ostermann, T. Wedi, et al., “Video coding with H.264/AVC: tools, performance, and complexity," IEEE Circuits and Systems Magazine, vol. 4, pp.7-28, 2004.

[8] Heejun Shim,Chong-Min Kyung, Selective Search Area Reuse Algorithm for Low External Memory Access Motion Estimation IEEE Transactions On Circuits And Systems For Video Technology, VOL. 19, NO. 7, July 2009

표 1. 움직임 추정 하드웨어의 Logic Utilization

Logic Utilization	Used	Available	Utilization
Combinational ALUTs	2,827	64,000	4%
Memory ALUTs	1,076	32,000	3%
Dedicated logic registers	4,523	64,000	3%
Total registers	4,523		
Total pins	467	744	63%

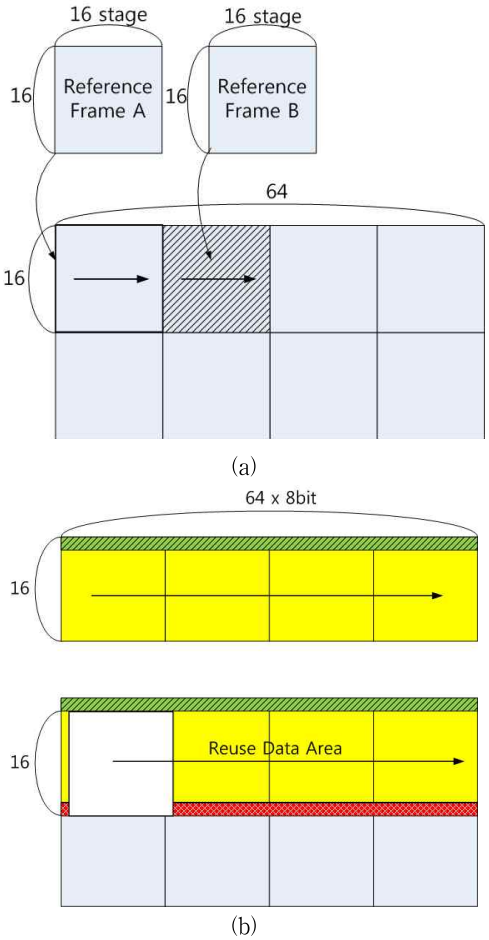

그림 8. Data Buffering (a) Current MB에 적용한 Data Buffering (b) Reference MB에 적용한 Data Buffering