碩士學位論文

개선된 MRME 알고리즘을 이용한 H.264 움직임 추정기 설계 및 FPGA 검증

濟州大學校 大學院

通信工學科

秦 君 宣

2004 年 12 月

개선된 MRME 알고리즘을 이용한 H.264 움직임 추정기 설계 및 FPGA 검증

指導敎授 林 載 允

秦 君 宣

이 論文을 工學 碩士學位 論文으로 提出함

2004 年 12 月

秦君宣의 工學 碩士學位 論文을 認准함

審査委員長 李 鎔 鶴 印

委 員 梁 斗 榮 印

委 員 林 載 允 印

濟州大學校 大學院

2004 年 12 月

H.264 motion estimation unit Design and FPGA verification using an enhanced MRME algorithm

Goon-sun Jin

(Supervised by professor Jea-yun Lim)

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Engineering

2004. 12.

This thesis has been examined and approved.

Thesis director, Yong-hak Lee, Prof. of Telecom. Eng.

Thesis director, Doo-Yeong Yang, Prof. of Telecom. Eng.

Thesis director, Jea-Yun Lim, Prof. of Telecom. Eng.

(Name and signature)

2004. 12. 20

Date

DEPARTMENT OF TELECOMMUNICATION ENGINEERING
GRADUATE SCHOOL
CHEJU NATIONAL UNIVERSITY

목 차

Abstract

H.264/AVC is newest video coding standard of the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group. The main goals of the H.264/AVC standardization effort have been enhanced compression performance and provision of a network-friendly video representation addressing conversational (video telephony) and "nonconversational" (storage, broadcast, or streaming) applications. H.264/AVC has achieved a significant improvement in rate-distortion efficiency relative to existing standards, such as H.263 and MPEG-4.

Usually, motion estimation is computationaliy expensive in a video encoder, and about 70% of the total encoding time is spent on this algoritnm. This heavy computational load limits the performance of the encoder in terms of encoding speed and power consumption. Hence, this paper presents an architectural enhancement to reduce the data load of the Multi-Resolution motion estimation. Our approch is based on eliminating unnecessary data load using memory reuse. New hardware architecture for integer-pel motion estimation(ME) dedicated to H.264/AVC is proposed. The proposed architecture supports all 7 modes (16x16, 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4) for variable block size ME. The features of our design are 2-D processing element(PE) array and SAD merging scheme. A pipelined and shared datapath architecture for motion estimation unit are designed to improve the system performance at the reduced hardware complexity. Proposed ME architecture can achive real-time processing for 352x288 @30Hz with search range of [-32, +31] in the horizontal and [-16, +15] in the vertical direction. In this paper, motion estimation unit are designed using Xilinx FPGA, coded in Verilog HDL, and simulated and verified using NC-Verilog

Ⅰ. 서론

H.264/AVC는 ITU-T 와 ISO가 함께 표준화 과정을 진행시켜 얻어낸 새로운 비디오 압축 표준으로 영상시스템과 관련된 모든 영역(DVD, digital camera, low bit rate wireless application 등)에의 적용이 가능한 압축기술로써 차세대 영상시스템을 주도할 수 있는 획기적인 코덱(codec)이라 할 수 있다. 파일 압축률이 높아 1Mbps 이하의 인터넷 망을 통해서 DVD 수준의 동영상을 전달하면서도 네트워크 자원을 덜 차지하는 강점이 있다.

H.264의 기본적인 개념 자체는 H.263과 MPEG-4와 유사하나, 세부적인 내부 구현에 있어 상당 부분 변경된 방식을 채택하고 있다. 움직임 추정 보상(Motion Estimation/ Compensation)의 경우를 살펴보면 H.264는 H.263이 모든 블록(Block)의 크기를 동일하게 하여 제공했던 것과는 달리 모양과 크기가 다른 블록들을 제공하고 1/4 pel 움직임 추정을 지원한다. 또한 H.264는 고성능 레벨에서 상호 연동을 증진시키는 등의 중요한 개선점과 강력한 에러 복구 기능 등을 제공한다. H.264는 부가된 특징과 기능을 통해 더 높은 압축 효율을 제공한다. 부가된 기능들은 일곱 개 모션 보상 모드, 다양한 레퍼런스 프레임, 효율적인 엔트로피 엔코딩, 그리고 예측적인 인트라 인코딩이 있어 저대역폭에서도 높은 화질을 기대할 수 있다.(A. Tamhankar 등 2003)

H.261이나 H.263같은 기존의 화상 압축 알고리즘은 고화질을 유지하기 위해 넓은 네트워크 대역폭이 필요로 한다. 따라서 채널이 열악하고 채널용량이 작아 압축율이 높아질 때, 압축율을 낮추기 위해 해상도를 줄이고 전반적인 이미지 감도를 높이는 것을 감수해야만 했다. 그러나 H.264 비디오 압축은 동일한 대역폭에서도 향상된 화질을 보장하며 종전 대역폭의 절반 속도에서도 동일한 화질을 제공한다. H.264 압축 알고리즘의 또다른 이점은 네트워크 에러시 향상된 성능을 볼 수 있다는 점이다. 즉, 네트워크 에러 때문에 화상 데이터에 손실이 있었을 때에도 화상이 완전히 끊어지는 대신 화질의 약화만 가져온다는 것이다.

따라서 H.264를 이용하여 영상통신 시스템을 구성한다면 기존 대역폭의 절반 정도에서 같은 정도의 화질을 기대할 수 있게 되었다. 또한 H.264는 고성능 레벨에서 상호 연동을 증진시키는 등의 중요한 개선점과 강력한 에러 복구 기능 등을 제공함으로써 네트워크 상태가 좋지 못할 때에서 화상회의가 계속될 수 있도록 해 준다.(Ajay K. Luthra 등, 2003)

H.264은 압축율 성능이 뛰어나므로 멀티미디어 어플리케이션이 저렴한 가격에서 보다 나은 품질을 제공할 수도 있으며, 또한 새로운 어플리케이션을 창출하는 견인차 역할도 수행할 것으로 기대된다.

H.264는 현재 우리나라 위성 DMB와 지상파 DMB의 동영상 코덱으로 채택되었으며 앞으로 HD DVD 플레이어, 디지털 TV, 디지털 캠코더, 휴대 이동 단말기에도 동영상 코덱 표준으로 채택될 가능성이 매우 높다. 이렇게 다양한 분야에서 사용될 H.264 영상 코덱을 ASIC화 한다면 정보가전 부품시장에서 상당한 점유율을 차지할 것으로 예상된다. 현재 H.264 영상 코덱을 DSP(Digital signal processor)등으로 구현하려는 시도는 많이 있지만 대부분 200~300MHz의 고주파에서 동작하는데 반해 ASIC으로 구현하였을 시에는 30MHz 이하에서도 실시간 영상 압축이 가능하여 전력 소모에 민감한 차량용 또는 이동 단말기에서 시스템 경쟁력을 매우 높일 수 있다. 또한 ASIC로 구현시에는 성능이 DSP로 구현한 시스템 보다 매우 안정적으로 동작하여 이동성이 많고 열악한 환경에서 동작해야하는 모바일 멀티미디어 단말에 특히 유리하다고 할 수 있다. 그리고 컨버전스화, 경박단소화를 만족해야하는 현대의 모바일 멀티미디어 단말에는 범용성을 위해 꼭 필요하지 않은 기능이 많이 내장된 범용 DSP보다는 이동 단말에 최적화된 ASIC이 절대적으로 유리하다.

본 논문에서는 H.264 영상 압축 ASIC에서 성능을 좌우하는 움직임 추정기를 구현하는데 있어 효율적인 구조를 갖고 성능 향상된 모듈을 설계하는데 목표를 둔다. 효율적인 설계를 위하여 움직임 추정을 위한 다양한 알고리즘을 분석하고 가장 적절한 알고리즘을 선택한다. 선택된 알고리즘의 핵심적인 기능을 분석하고, 이를 구현하기 위해 적합한 움직임 추정기 전체 구조를 제안한다. 그리고 전체 구조에 속하는 여러 중요 모듈 설계 구조에 대한 연구와 움직임 추정 알

고리즘을 가장 적합하게 구현할 수 있는 구조에 대해 심도있게 분석한다. 또한 움직임 추정기의 내부 구조와 이를 구현하기 위한 설계 방법 및 테스트 환경을 살펴본다. 보드레벨에서의 검증은 EISC(extensible instruction set computing) 방식의 SE3208 코어 MCU 연결하여 FPGA(field programmable gate array)로 검증하고 그 결과를 비교・검토한다.

본 논문의 구성을 살펴보면, Ⅱ장에서는 움직임 추정을 위한 다양한 알고리즘에 대해서 설명을 기술한다. Ⅲ장에서는 H.264 움직임 추정기 구현을 위한 내부 구조를 제시하고 설계방법을 설명한다. Ⅳ장에서는 H.264 움직임 추정기를 Verilog HDL로 기술하여 시뮬레이션 결과 및 합성 결과를 제시하며 FPGA 상에서 구현하여 검증된 결과를 제시한다. 또한 설계된 움직임 추정기를 기존의 움직임 추정기와 비교하여 어느정도의 성능 향상이 있는지 알아본다, 마지막으로 Ⅴ장에서는 본 논문의 결론을 맺는다.

Ⅱ. 움직임 추정 알고리즘 및 비교

1. 동영상 시스템에서의 움직임 추정

동영상 시스템에서의 압축 부호화를 위해서는 동영상에 존재하는 4가지 중복성, 즉 신호 성분간에 존재하는 중복성, 화면 내에 존재하는 공간적 중복성, 화면간에 존재하는 중복성, 그리고 데이터의 통계적 발생 확률에 존재하는 통계적 중복성을 효과적으로 제거해야 한다. 신호 성분간에 존재하는 중복성은 휘도 신호(luminance : Y)와 색차 신호(chrominance : Cb, Cr)의 비율을 2:1로 정의함으로서 제거하며, 통계적 중복성은 데이터의 발생 확률 분포를 활용하여 평균 비트수를 줄이는 가변장 부호화에 의해 제거한다. 화면 내에 존재하는 공간적 중복성은 변환부호화와 양자화 과정을 통해서 제거하고 화면간에 존재하는 시간적 중복성은 움직임 추정/보상(motion estimation/compensation : MV/ MC) 기법에 의해 제거된다.

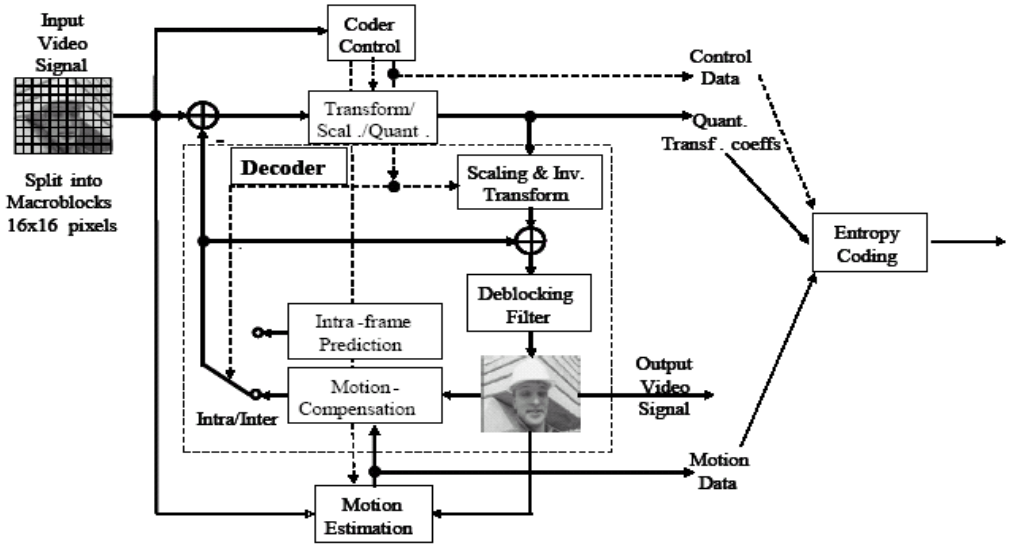

Fig. 1. Block diagram of moving picture encoder

Fig. 1은 전형적인 동영상 부호화기의 블록도를 나타낸 것이다. 주요 블록은 주파수 변환단(Transform unit), 움직임 추정단(motion estimation unit), 움직임 보상단(motion compensation unit), 허프만 부호화단(entropy coding unit)이다.

동영상 부호화기의 압축 과정에서 시간적 중복성을 제거하기 위해 이전 프레임의 데이터를 이용하여 움직임 추정 및 보상을 수행하고, 이때 추정된 움직임 벡터(motion vector : MV)에 의해 보상된 영상과 원 영상과의 차 신호를 부호화하여 전송하게 된다. 이 방법은 동영상 부호화에서 가장 높은 압축률을 가져오지만 많은 계산량으로 인하여 부호화기의 전체 성능에 결정적인 영향을 미치게 된다.

Fig 2는 움직임 추정 과정을 간략하게 나타낸 그림이다. 움직임 추정은 이전 화면에서 현재 화면을 예측하는 방법이므로 이 과정은 기준블럭과 이전 화면 블럭간의 화소값 차이를 계산하여 그 값이 가장 작은 블럭을 택하는 방식으로 진행된다. 이때 행해지는 연산은 보통 회로 구현에 있어 용이한 SAD(sum absolute difference)가 사용된다. R은 참조 블록이며 S는 탐색 영역 블록을 말한다.

$$SAD(u,v) = \sum_{i=1}^{N} \sum_{J=1}^{M} |R(i,j) - S(i+u, j+v)| \quad (1)$$

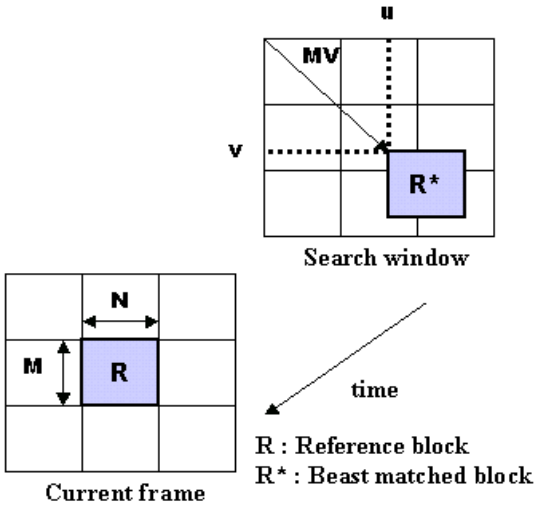

Fig. 2. Motion estimation

2. 전역탐색 알고리즘

연속하는 영상의 움직임 벡터를 추정하기 위한 많은 방법들이 연구되어왔는데 크게 픽셀 단위로 움직임을 추정하는 화소 순환 방식(pixel recursive algorithm : PRA)과 블록 단위로 움직임을 추정하는 블록 정합방식(block matching algorithm : BMA)으로 대별된다. 픽셀 단위로 움직임을 추정하면 그 어떤 방법보다도 정확한 추정이 가능하지만 계산량이 너무 많고 움직임 벡터 전송시 각 픽셀의 움직임 정보를 보내야 하므로 전송해야하는 정보량이 너무 많아져 비효율적이다. 반면 블록 정합방식은 움직임 추정 방식이 비교적 간단하고 하드웨어 구현이 용이하며 예측 효율과 추정의 정확도 등을 고려했을 경우 전체적으로 좋은 특성을 보인다.

전역탐색(full search block matching) 알고리즘은 일정한 탐색 영역에서 모든 후보 블록을 순차적으로 비교하여 가장 유사한 블록을 찾는 방법이지만 많은 계산량 때문에 영상 전화, 영상회의, 고화질 TV, 주문형 비디오와 같은 실시간 비디오 코딩 응용 분야 및 소프트웨어 구현에 많은 어려움이 따른다.

3. Three STEP 알고리즘

블록 매칭 알고리즘에서는 전역탐색(full search)방법이 주로 사용된다. 그러나 전역탐색 방법에서 예측된 영상의 화질은 우수하지만 연산량이 너무 많아 실시간 시스템 구현에 어려움이 있기 때문에, Three STEP, Four-STEP 고속 알고리즘이 연구되어 오고 있다.

Three STEP 알고리즘은 9개의 탐색 포인트를 정하고 탐색 간격을 줄여 가면서 움직임 벡터를 찾는 간단한 방법이다. Fig 3은 이 과정을 설명하는 그림이다. 여기에서 탐색 간격은 처음에는 3픽셀, 두 번째 단계에서는 2픽셀, 마지막 단계에서는 1픽셀 간격으로 줄여간다. 탐색 순서는 다음과 같다.

1) 탐색 간격을 3픽셀로하여 최소가 되는 SAD를 찾는다.

2) 1)에서 찾은 점을 중심으로 탐색 간격을 2픽셀로 줄여서 최소가 되는 포인트를 찾는다.

3) 탐색 간격을 한 샘플 간격이 될 때 까지 반복한다.

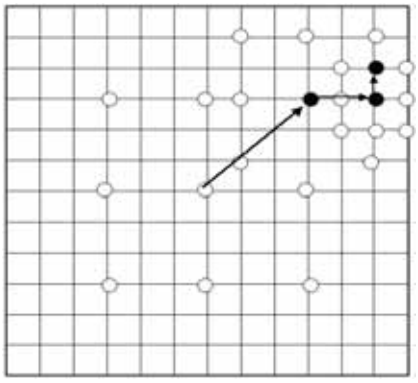

Fig. 3. Three step algorithm

4. 계층적 탐색 알고리즘

계층적 탐색 알고리즘은 위에서 설명한 전역 탐색 알고리즘과 Three Step 알고리즘의 단점을 보완한 새로운 움직임 추정 알고리즘이며 새롭게 각광받고 있는 알고리즘이다. 이 알고리즘이 주목 받는 이유는 Three Step 알고리즘과 같이 계산량은 적으면서도 전역탐색 알고리즘에 버금가는 움직임 추정 성능을 나타내기 때문이다.

계층적 탐색 알고리즘의 움직임 추정 방식은 Fig. 4와 같이 세 단계로 나누어지는데 그 기준은 해상도에 따른 분류이다. 프레임 크기에 따른 원래의 탐색 영역은 해상도가 가장 큰 단계이며 다음 단계는 첫 단계의 영역을 2:1로 downsampling 하여 얻어지며 마지막 단계의 해상도는 중간 단계의 영역을 다시

2:1 downsampling하여 얻어진다. 이렇게 얻어진 마지막 단계 해상도에서부터 움직임 추정을 수행해 나가게 된다. 가장 낮은 해상도에서는 탐색 영역이 크지 않으므로 전역 탐색을 수행하여 최소가 되는 움직임 벡터를 찾고 중간 단계 영역에서는 이 움직임 벡터를 중심으로 움직임 추정을 수행한다. 이렇게 얻어진 중간 단계의 움직임 벡터를 중심으로 가장 높은 해상도의 탐색 영역에서 움직임 추정을 수행한다. Fig. 4는 위의 계층적 탐색 알고리즘을 알기 쉽게 나타낸 그림이다. 세 단계에 걸쳐 찾은 움직임 벡터는 전역 탐색 알고리즘을 이용하여 얻어진 움직임 벡터와 거의 일치하며 계산량은 Three Step 알고리즘보다는 조금 많은 정도인 것을 여러 논문에서 확인할 수 있다.

이러한 이유로 본 논문에서도 계층적 탐색 알고리즘을 기본 원리로 하여 새롭게 고안된 MRME(multi resolution motion estimation)알고리즘을 움직임 추정기 설계에 적용하고자 한다.

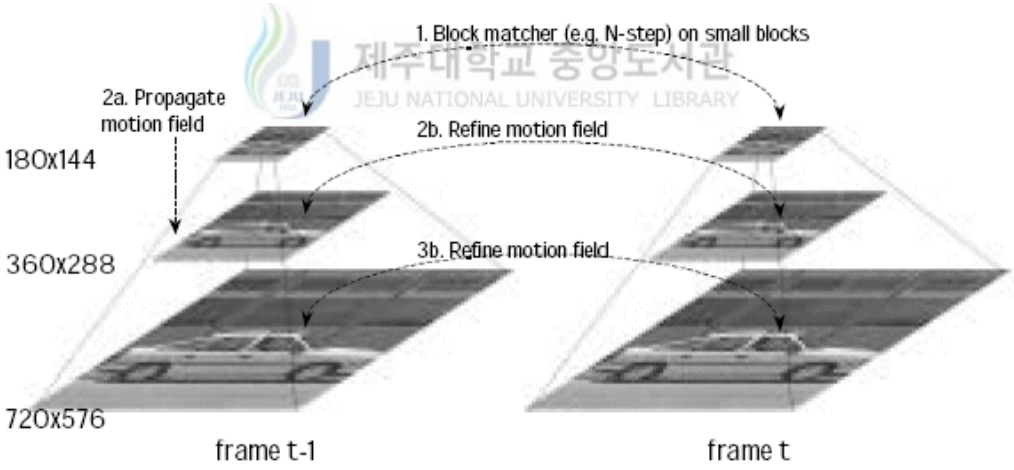

Fig. 4. Hierachical motion estimation

Ⅲ. 개선된 MRME 기반 움직임 추정기 설계 구조

1. 개선된 MRME 알고리즘 분석

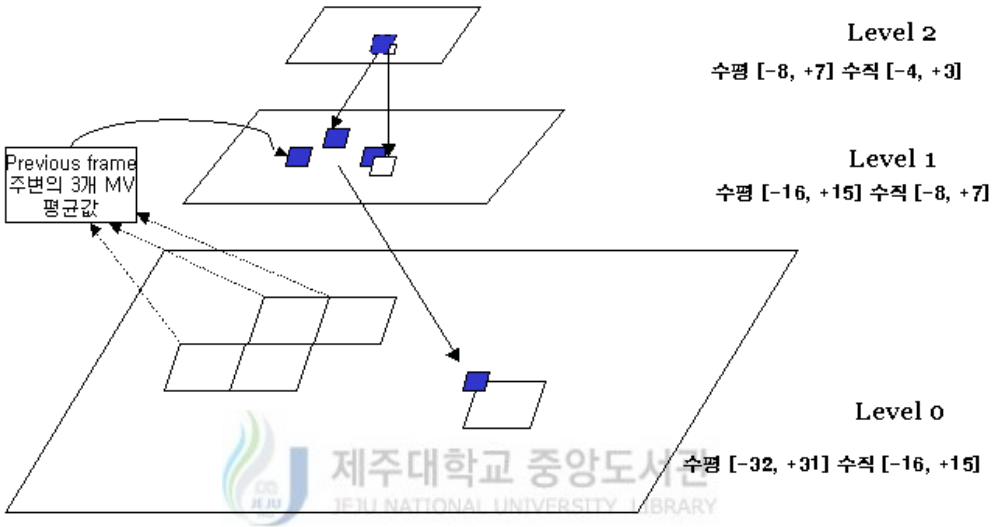

Fig. 5. Enhanced MRME algorithm

본 논문에서는 (Jae Hun Lee 등 2001)에서 제안되었던 다해상도 다중후보 탐색 움직임 추정 알고리즘 (multi-resolution multiple candidate Search: MRMCS)을 개선하여 H.264/AVC 하드웨어 구현이 용이하며 다양한 블록 크기에 대응하도록 확장한 알고리즘을 이용한다. Fig. 5는 제안하는 움직임 추정 알고리즘을 나타낸다. 레벨2에서는 움직임 추정기로의 입력 영상을 수평, 수직 방향으로 4:1로 샘플링(sampling)한 영상을 이용하여 [-2, +1] 탐색영역에서 4x4 블록 단위로 움직임 추정을 수행한다. 레벨2에서 탐색을 수행하여 차의 절대값(sum absolute difference)이 최소가 되는 2점을 구한다. 그리고 인접한 매크로블럭의 움직임 벡터들의 중간값을 이용하여 한 점을 구한다. 이렇게 구해진 3개의 점을 이용한다. 레벨1에서는 원래의 영상을 수평, 수직으로 2:1로 샘플링(sampling)한 영상에서 위

의 3개의 탐색 중심점을 중심으로 8x8 블록 단위로 [-2, +1]의 부분 탐색을 수행한다. 탐색 수행 후 차의 절대값이 최소가 되는 하나의 점을 선택하여 하위 레벨의 탐색 중심점으로 이용한다.

레벨0에서는 원래의 영상에서 16x16 매크로 블록 단위로 수평, 수직 [-2, +1]의 부분 탐색을 수행한다. H.264/AVC의 다양한 블록 크기 (16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4)를 지원하기 위해 4x4 블록 단위로 정합 기준을 구하고 이를 더하여 4x4 블록보다 더 큰 블록에 대한 정합 기준을 계산하는 방식을 이용한다.

Fig. 6는 레벨2에서의 탐색 영역을 나타낸 그림이다. 영상의 탐색 영역은 보통 영상 포맷에 따라 결정되는데, SD급 및 VGA 영상은 수평 [-64, +63], 수직 [-32, +31]의 탐색 영역을 갖는다. 본 논문의 움직임 추정기는 CIF급 영상 포맷을 목표로 설계하므로 수평 [-32, +31], 수직 [-16, +15]의 탐색 영역으로 결정하였다.

레벨0에서의 탐색 영역은 수평 방향으로 [-32,+31], 수직 방향으로 [16, +15]이므로 4:1 downsampling된 레벨2의 탐색 영역은 수평 방향으로 [-8, +7], 수직 방향으로 [-4, +3]이 된다. 수직 방향의 탐색 영역이 수평 방향의 탐색 영역의 1/2 이 되는 것은 영상의 통계학적인 특성을 이용한 것인데 일반적으로 영상은 수직 방향으로의 움직임 보다 수평 방향으로의 움직임이 대부분이기 때문이다.

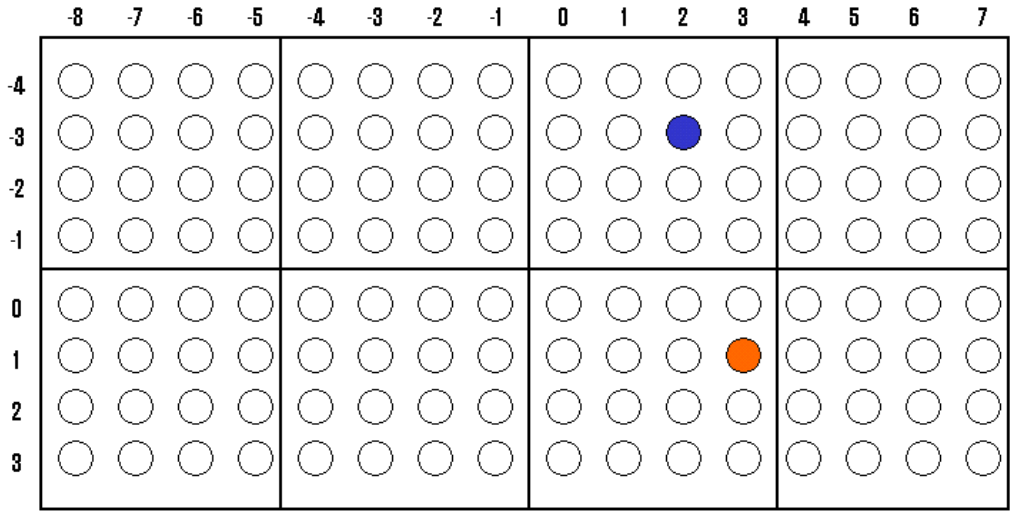

Fig. 6. Search range in the level2

레벨2에서의 기준 블록은 4x4 매크로블럭이며 PE_ARRAY 블록의 기준 블록은 8x8 이므로 단일 클럭에서 동시에 4개의 움직임 벡터를 찾게 된다. 이러한 고속 움직임 탐색을 통해 레벨2에서는 2개의 최소가 되는 움직임 벡터를 찾게 된다.

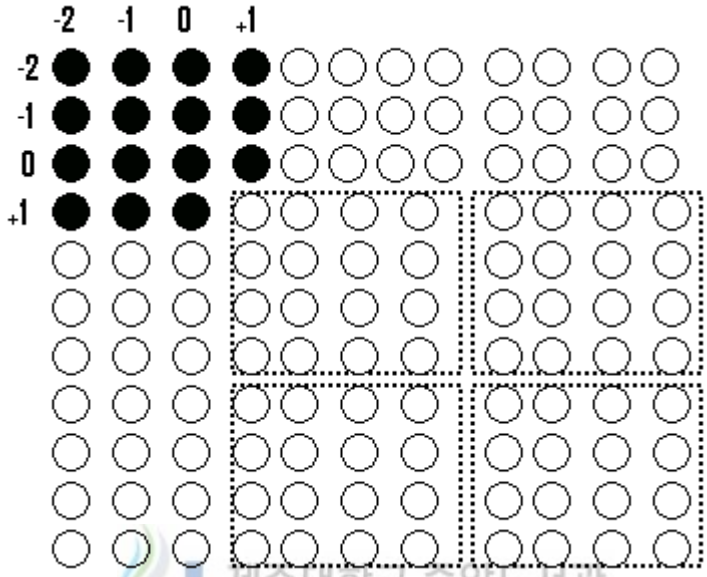

Fig. 7. 11x11 search range

레벨1에서의 탐색 영역은 레벨0 탐색 영역의 반 이므로 수평 방향으로 [-16, +15], 수직 방향으로 [-8, +7]이 된다. Fig. 7는 레벨1에서 8x8 메크로 블록을 이용하여 [-2, +1] 영역의 움직임 탐색을 수행하는 것을 나타낸다. 레벨2에서 계산된 3개의 움직임 벡터를 중심으로 움직임 탐색을 수행하게 된다.

레벨0에서는 수평 [-32, +31], 수직 방향으로 [-16, +15] 탐색 영역에서 16x16 메크로 블록을 이용하여 [-2, +1] 탐색을 수행하게 된다. Fig. 7에서 8x8 매크로 블록 대신 16x16 매크로 블록만 대체하면 되므로 그림은 생략 하였다. 레벨0에서 계산된 최소의 움직임 벡터가 우리가 원하는 최종의 움직임 벡터이다. 16x16 매크로 블럭을 이용하므로 단일 클럭에서 SAD 값을 계산 할 수 없고 4개의 8x8 매크로 블록에 대한 저장 값을 이용하여 움직임 벡터를 찾는다.

2. 제안하는 움직임 추정기 전체 구조

본 논문에서 제안하는 움직임 추정기 전체 구조는 Fig. 8 과 같다. 주요 블록으로는 AMBA Bus와 인터페이스 되는 AMBA I/F, AMBA Bus를 통해 전송되는 영상 데이터를 3개의 레벨로 분리 시키는 해상도 변환기(resolution converter), 영상 데이터를 저장하는 RAM 배열기(ram_array), RAM을 제어하는 메모리 제어기(ram controller), PE_ARRAY 블록의 입력값을 8x8 형태로 변환하는 레지스터 배열기(reg_array), 영상데이터의 차의 절대값을 구하는 PE_ARRAY, 각 SAD 합을 비교하여 최소가 되는 움직임 벡터를 추출하는 비교단(compare unit), 움직임 추정기 전체 제어를 담당하는 전체 제어기(MRME controller) 이다.

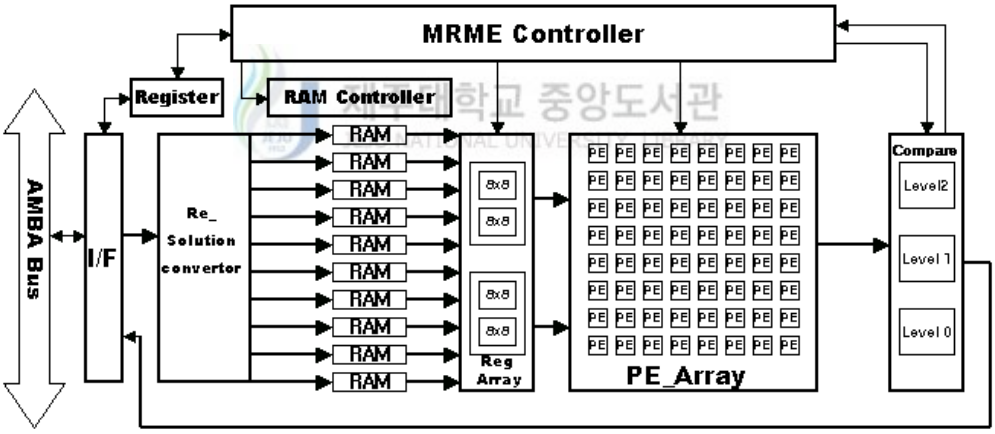

Fig. 8. Motion Estimation hardware architecture

3. 주요 블록 설계 구조

(1) AMBA I/F unit 구조 설계

Fig. 9은 AHB 버스와 인터페이스 하기 위한 정보 및 제어 신호를 나타낸 것이다. AHB 버스와 인터페이스 되는 IP의 데이터값을 전송할 어드레스 정보 신호인 HADDT[31:0]가 클럭에 동기되어 전송된다. 그리고 HADDR[31:0]과 HWDATA[31:0]를 받아 들이기 위한 control 신호가 함께 전송된다. AHB 버스에 인터페이스된 IP는 HREADY 신호가 HIGH 일때 HRDATA[31:0] 신호를 입력 받게 되는 것이다.

Fig. 10는 Fig. 9의 단순 전송에서 다수의 데이터를 전송하는 타이밍도를 나타내는 그림이다. Master에서 연속적으로 어드레스와 제어 신호, 데이터 값이 전송되어질때 AHB 인터페이스된 IP가 어떤 타이밍도를 만족하면서 데이터값을 받아들이는지 보여주는 그림이다.

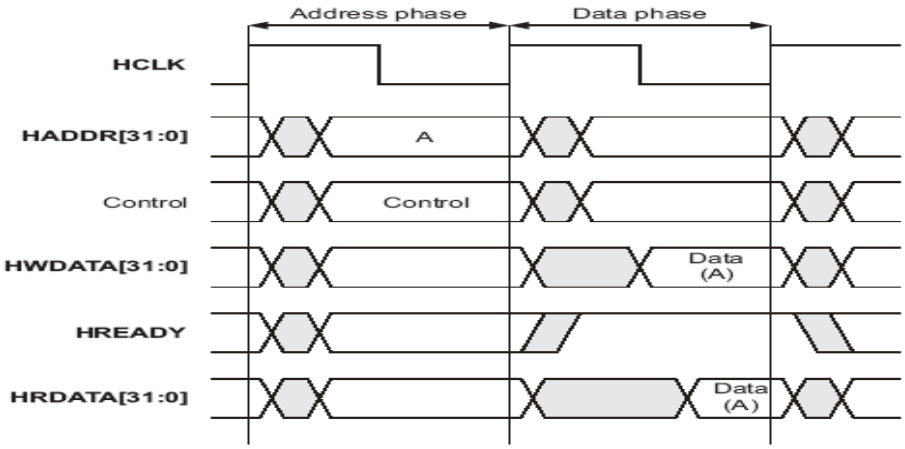

Fig. 9. Simple transfer

Fig. 11는 전송타입을 잘 설명해주는 타이밍도이다. 첫 전송에서 HTRANS[1:0]은 NONSEQ가 되는 것은 당연하다. 그리고 다음 클럭에서 바로 받아 들일 수 없으므로 HTRANS 신호는 BUSY 신호를 출력함으로써 IP에게 알린다. 그림에서와 같이 HBURST[2:0] 신호는 INCR이다. 여기서 INCR은 증가(increment)의 약자로서 HADDR[31:0] 신호가 증가되는 주소값을 가짐을 의미한다. 그림에서 알 수 있듯이 AHB 버스와의 인터페이스는 아주 단순하면서도 정교하게 신호를 주고 받으면서 안정적으로 데이터값을 입출력하게 설계되어 있음을 알 수 있다.

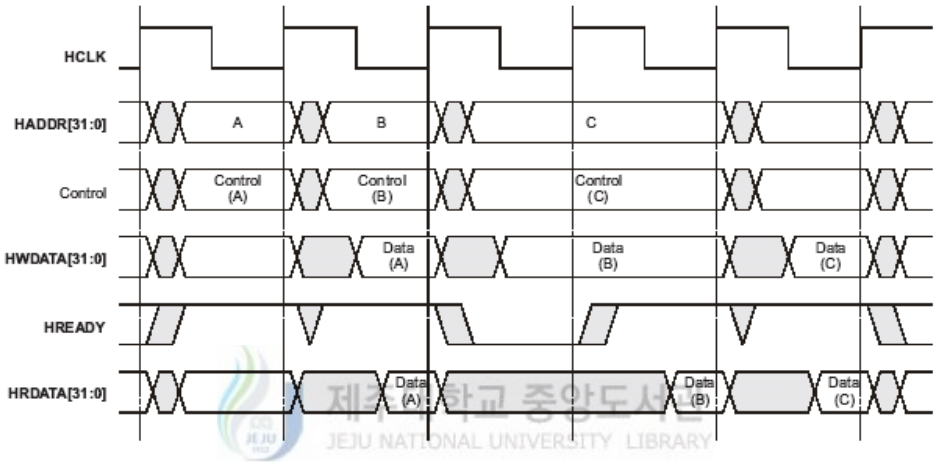

Fig. 10. Multiple transfers

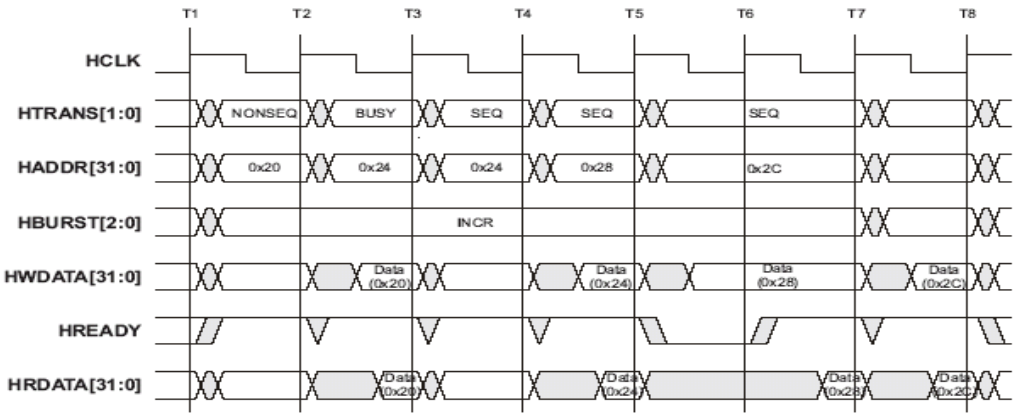

Fig. 11 Transfer with INCR type

(2) 해상도 변환기 구조 설계

해상도 변환기(resolution converter)는 AMBA I/F unit으로부터 전달된 참조 영상 및 탐색 영역 영상을 3개의 레벨로 분리 시키는 역할을 수행한다. 레벨0에서의 영상이 AMBA I/F unit으로 전달된 영상이며 레벨1의 영상 데이터는 원래의 영상을 2:1로 샘플링하여 얻는다. 마찬가지로 레벨2에서의 영상의 레벨1 영상을 2:1로 샘플링하여 얻는다.

Fig. 12은 해상도 변환기의 대략적 구조를 나타낸다. prev_resol_data는 탐색 영역의 32비트 데이터 이며, cur_resol_data는 참조 영상 데이터이다. 입력 데이터는 32비트 버퍼로 입력되며 resol_controller와 resol_cnt에 의해서 각 레벨을 위한 영상 데이터를 얻는다.

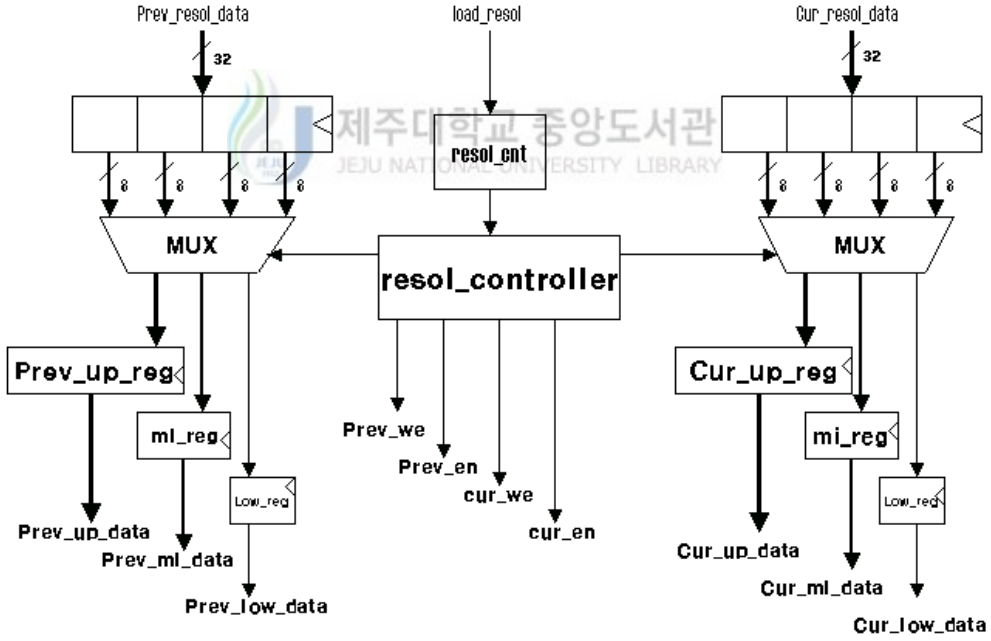

Fig. 12. Resolution converter architecture

(3) 메모리 제어기 구조 설계

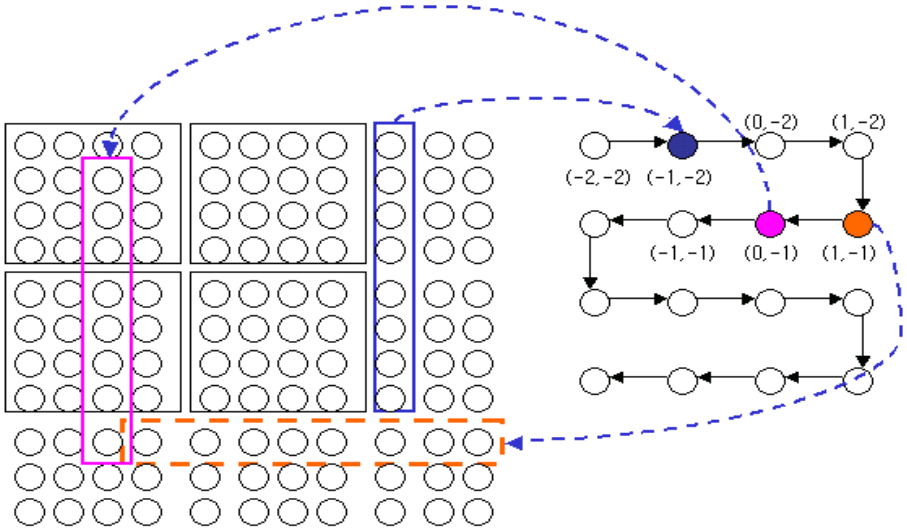

Fig. 13. Concept of memory reuse

메모리 제어기는 탐색 영역 데이터와 참조 영상 데이터를 저장하는 RAM을 위한 제어기이다. 제어 신호와 어드레스 값, 데이터 값등을 발생시켜 RAM에 데이터를 저장하기도 하고 저장된 데이터를 다시 읽어 오기도 한다. 영상 시스템은 특성상 메모리에 데이터를 저장하고 읽어 오는 기능을 수행할 때가 빈번하며 데이터양 또한 대용량일 경우가 많다. 특히 영상 시스템에서 움직임 추정기는 여러 프레임 간의 움직임 추정을 수행하므로 저장하고 읽어오는 데이터의 이동이 가장 빈번하다. 이러한 움직임 추정기에서 메모리 컨트롤러를 최적화 하지 않는다면 움직임 벡터를 계산하는 것 보다 계산할 영상 데이터를 읽어 오는데 필요한 클럭이 과도하게 지연되여 움직임 추정기의 성능을 상당 부분 저하시키게 된다.

따라서 본 논문에서는 움직임 추정기 성능 향상을 위한 메모리 재사용(memory reuse) 방법을 제안하고 이를 구현 하고자 한다. Fig. 13은 메모리 재사용 방법을 이용하여 [-2, +1] 탐색에 해당하는 16개 탐색 순서와 탐색을 위한 데이터 흐름을 나타낸 것이다. 메모리 재사용을 위해 4가지의 입력 상태가 있는데 다음과 같다.

먼저 8x8 블록의 입력을 모두다 받아 들이는 주소 증가(inc_add) 상태인데 Fig. 13에서 처음 8x8 블록의 데이터에 해당하는 사각형 그림이다. 다음은 오른쪽으로 이동하면서 행(row)방향으로 8개의 입력값을 받아 들이는 오른쪽_주소(right_add) 상태인데, 좌측으로 한 픽셀씩 이동하면서 (-1, -2), (0, -2), (1, -2) 위치에서의 움직임 벡터를 찾게된다. 다음은 (1, -1) 탐색점의 움직임 벡터를 찾게 되는데 Fig. 13에서 열(column)방향으로 8개의 입력값을 받아 들이는 아래쪽_주소(dw_add) 상태이다. 또한 왼쪽으로 이동하면서 행(row)방향으로 8개의 입력값을 받아 들이는 왼쪽_주소(left_add) 상태가 있는데 우측으로 한 픽셀씩 이동하면서 (0, -1), (-1, -1), (-2, -1) 위치에서의 움직임 벡터를 찾게된다.

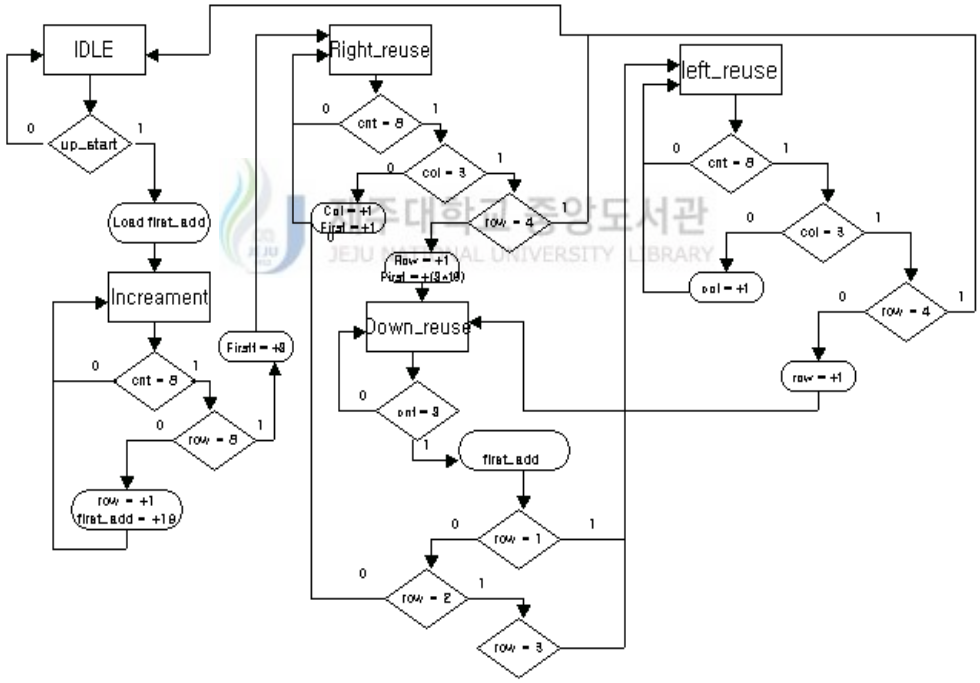

Fig. 14. State machine for memory controller

Fig. 14.은 메모리 재사용 구현을 위한 SM(state machine) 차트이다. 앞에서 설명하였듯이 메모리 재사용을 위한 증가_주소(increment), 오른쪽_주소

(right_reuse), 아래쪽_주소(down_reuse), 왼쪽_주소(left_reuse)의 대표적인 상태가 있다. 구현은 카운터를 이용하여 각 상태마다 입력값을 세어 다음 상태로 천이하게 되어 있다. 그림에서 보면 처음 상태도는 IDLE 상태에 있다가 start 신호가 가해지면 입력값을 8x8 만큼 받아 들이게 된다. 행과 열 방향으로 입력값을 모두 받아 들이게 되면 다음 상태인 오른쪽_주소(right_reuse) 상태로 천이하게 된다. 여기서도 8개의 입력값을 받아들이면 오른쪽으로 한 픽셀 이동하여 새로운 시작 주소값을 계산하고 이 주소값을 바탕으로 8개의 입력 영상 데이터 값을 받아들인다. 3번의 오른쪽 이동이 이루어지게 되면 다음 상태는 아래쪽_주소(down_reuse) 상태로 천이한다. 아래쪽_주소(down_reuse) 상태는 8x8 매크로블럭의 제일 마지막 행의 데이터 값을 새롭게 받아들이는 상태이므로 열(column) 방향으로 8개의 데이터를 받아들이고 다음 상태로 천이하게 된다. 왼쪽_주소(left_reuse) 상태는 왼쪽으로 3번 이동하면서 데이터를 받아들이는 상태이다.

(4) PE_ARRAY 설계 구조

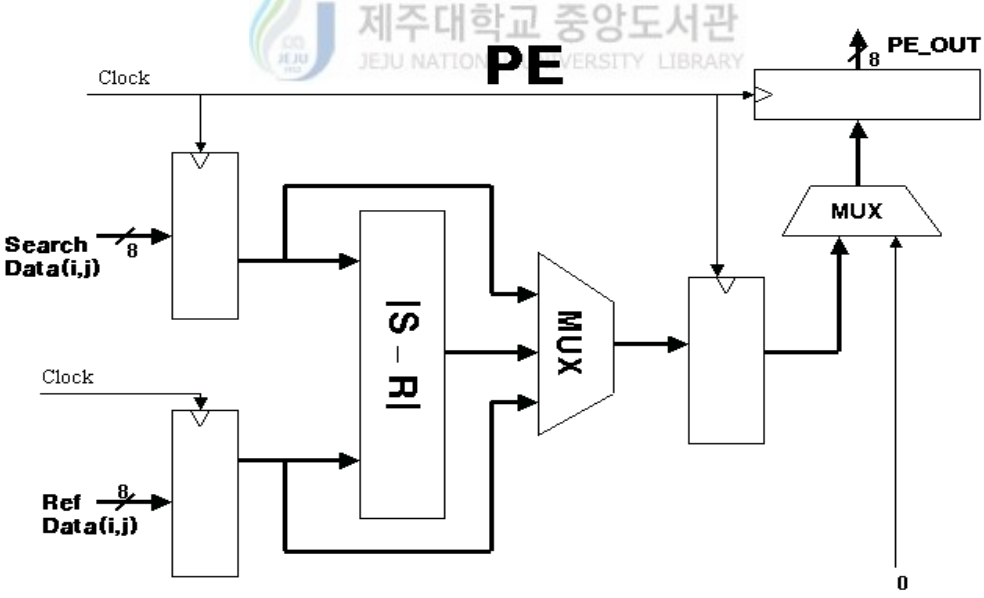

Fig. 15. Processing element architecture

Fig. 15.은 PE(processing element)를 나타낸 것이다. PE는 탐색 영역 데이터(search data)와 참조 영상 데이터(ref data)의 차의 절대값을 구하는 기본 장치이다. 그림에서 보는 것과 같이 탐색 영역 데이터와 참조 영상 데이터는 입력으로 들어와 하나의 레지스터에 잠시 저장되었다가 다음 클럭에서 차의 절대값을 구하는 부분으로 들어가 차의 절대값을 구하게 된다. 두개의 입력 데이터 중 하나가 "0"이라면 차의 절대값을 계산할 필요없이 "0"이 아닌 값을 출력 시키면 되므로 출력 레지스터 전단의 MUX는 이러한 기능을 수행해 준다.

Fig. 16.는 PE_ARRAY unit를 나타낸 것이다. 그림에서 보는 것과 같이 전에 설명한 PE 64가 배열(array) 형태로 내장되어 있다. 입력값으로는 레지스터 배열기(reg_array)에서 출력된 8x8 매크로블럭 단위의 탐색 영역 데이터와 참조 영상 데이터 128개 이다. 단일 클럭에서 계산된 64개의 차의 절대값의 합은 4부분으로 나뉜 각 블록의 합으로 출력된다. 즉, 출력은 PE 16가 모인 pe_sad_sum0, pe_sad_sum1, 2, 3 이다.

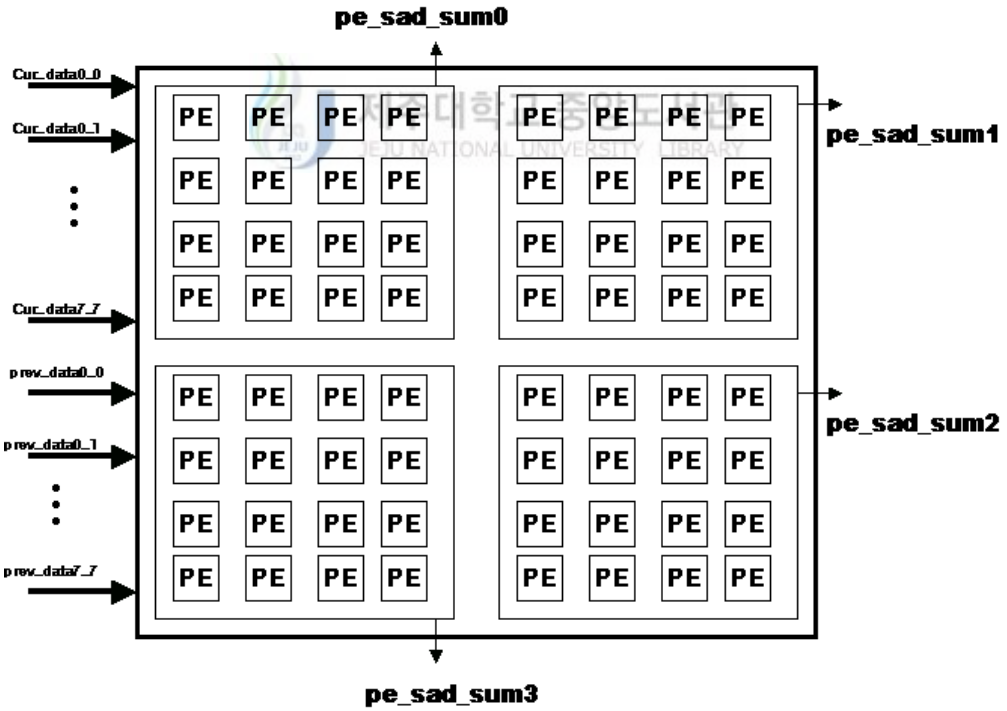

Fig. 16. PE_Array architecture

(5) 레지스터 배열기 구조 설계

레지스터 배열기는 PE_ARRAY에 8x8 매크로블럭 데이터를 전달하기 위한 중간 버퍼 역할을 하는 모듈로서 RAM으로부터 읽혀진 탐색 영역 데이터와 참조 영상 데이터를 메모리 재사용 방법에 따라 8x8 매크로 블록 데이터로 만들어준다. Fig. 17은 reg_array unit을 나타낸 그림으로 크게 controller, MUX, DEMUX, register 부분으로 나눌 수 있다. 레지스터는 first_reg와 two_reg로 나뉘는데 각각 8비트의 데이터 크기를 갖는 64개 array로 구성된다. 입력 신호로 inc_add_out가 들어오면 first_reg에 데이터값을 저장시키며 다음 신호인 lf_add_out 신호가 입력되면 first_reg 데이터값을 왼쪽으로 한번 쉬프트된 값을 two_reg에 저장시키며 비어있는 8개의 register에는 새로 입력되는 데이터값을 저장한다. 앞에 설명하였던 메모리 재사용 방법 단계에 따라 레지스터 이동을 제어한다.

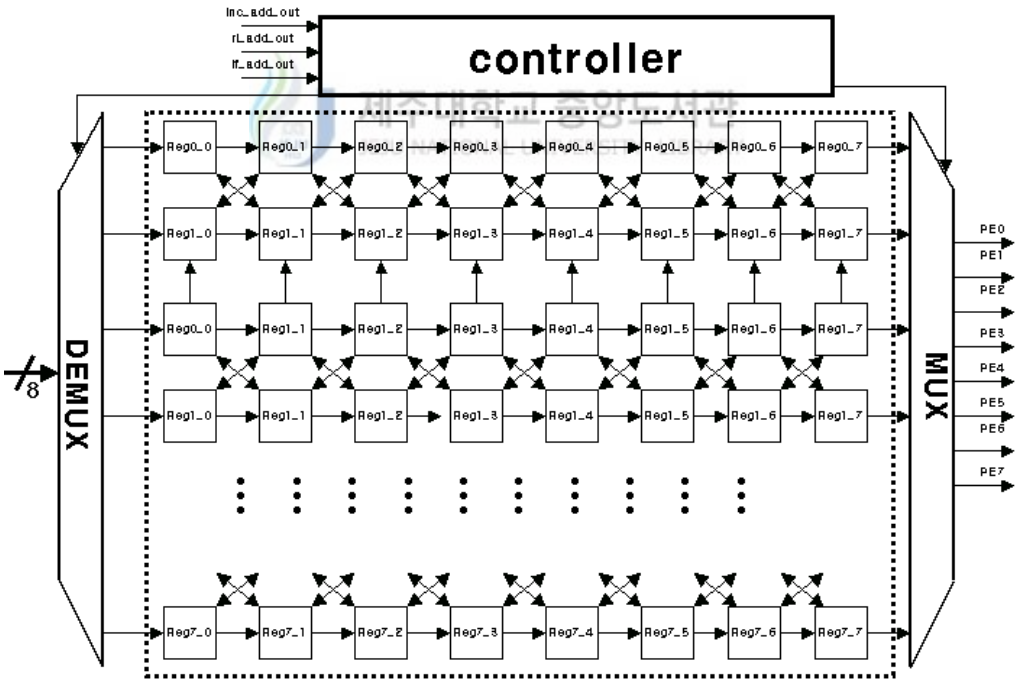

Fig. 17. REG_Array architecture

(6) 비교단 구조 설계

비교단은 PE_ARRAY에서 출력된 SAD 합을 이용하여 최소가 되는 움직임 벡터를 찾는 모듈이다. 본 논문에서의 움직임 벡터 발생기는 각 레벨에서의 비교단이 존재하는데 Fig. 18은 레벨0에서의 비교단 구조도를 나타낸 것이다. 레벨0에서의 움직임 추정은 16x16 매크로블럭 단위로 수행하므로 다른 레벨에서의 비교단보다 더 복잡하다. 우선 PE_ARRAY에서는 8x8 단위로 SAD값을 계산하므로 그림에서와 같이 temp0 ~ temp15에 저장을 한 후 비교를 수행하게 된다. 그림에서 보는 바와 같이 입력으로 sum0~3의 sad 합의 값이 입력되며 이 입력값을 합하여 4개의 부분으로 switching 하여 준다. [-2, +1] 영역에서의 모든 sad 합이 입력 완료되면 temp에 저장된 값을 이용하여 비교를 수행하게 된다. [-2 ,+1] 각 픽셀 위치에서의 16x16 매크로 블록을 비교하게 되면 8개의 최소가 되는 합이 출력되며 이 값은 다시 비교기로 가서 4개의 최소가 되는 합을 추출하게 된다. 이러한 과정을 통해 최종으로 최소가 되는 합을 구하게 되며 그 위치에서의 움직임 벡터를 출력하게 된다.

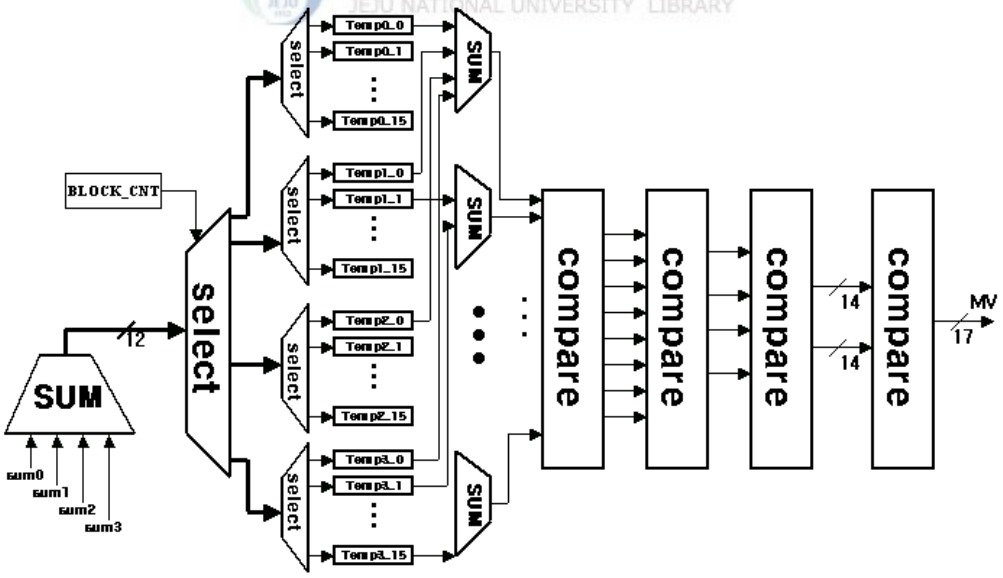

Fig. 18. Compare unit architecture

Ⅳ. 움직임 추정기 구현, 기능 검증 및 성능 평가

1. 움직임 추정기 모듈 설계

(1) AMBA I/F 모듈

AMBA I/F 모듈은 내장형 MCU와 움직임 추정기간의 AMBA 버스를 통한 통신 인터페이스를 위한 블록이다. AMBA I/F동작은 크게 어드레스 모드, 전송 모드, 에러 모드로 나눌 수 있다. 어드레스 모드에서 움직임 추정기 모듈의 읽거나 쓸 주소를 보내며, 전송 모드에서는 데이터를 주고 받는다. 여기서 AMBA I/F 통한 움직임 추정기는 슬레이브에 해당하며 마스터는 내장형 MCU 및 DMA가 된다.

Fig. 19은 AMBA I/F모듈의 입출력 포트를 나타낸 그림이다. hburst[2:0], htrans[1:0], hsize[1:0]등과 같이 통신을 위한 포트가 정의되며 읽고 쓰기 위한 제

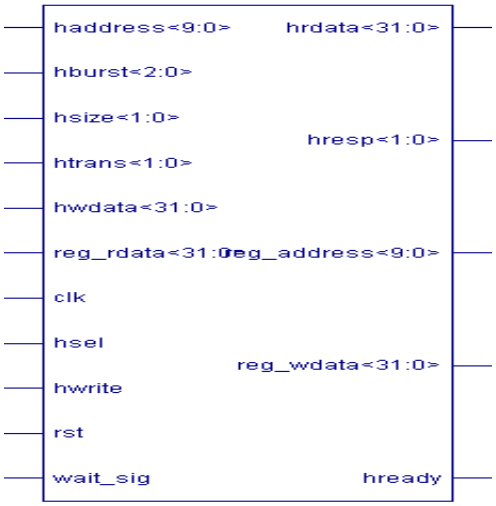

Fig. 19. Block diagram of AMBA interface unit

어 신호인 hwrite, 데이터 신호 hwdata[31:0], 어드레스 신호 haddress[9:0] 등의 신호가 있다. Fig. 20.는 AMBA I/F모듈을 Verilog HDL로 코딩하고 NC-Verilog tool을 사용하여 시뮬레이션한 결과이다. AMBA 인터페이스를 위한 제어신호가 적절히 발생하여 움직임 추정기로 어드레스와 데이터가 추출되는 것을 검증하였다.

내장형 MCU는 RISC의 스케줄러에 의해 AMBA 버스를 통해 움직임 추정기에 데이터 전송할 디바이스 선정, 디바이스 시작 어드레스, 전송 길이, 전송 방향 등을 보낸다.

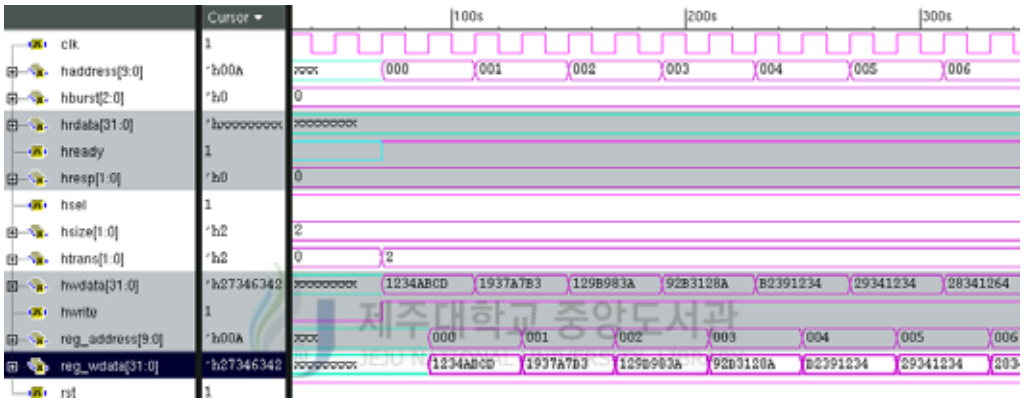

Fig. 20. Simulation result of AMBA I/F

버스 중재기(BUS Arbiter)는 버스 요청 신호를 받아서 우선순위에 따라 허가(Grant) 신호를 보낸다. 이 신호를 받으면 내장형 MCU는 버스 사용권을 가지게 된다. 버스 디코더(BUS Decoder)는 어드레스를 분석하여 데이터를 전송할 디바이스를 선택(Chip select)하는 역할을 수행한다. Fig. 21.은 마스터와 슬레이브, 버스 중재기, 버스 디코더의 기능 블록도를 나타낸 그림이다.

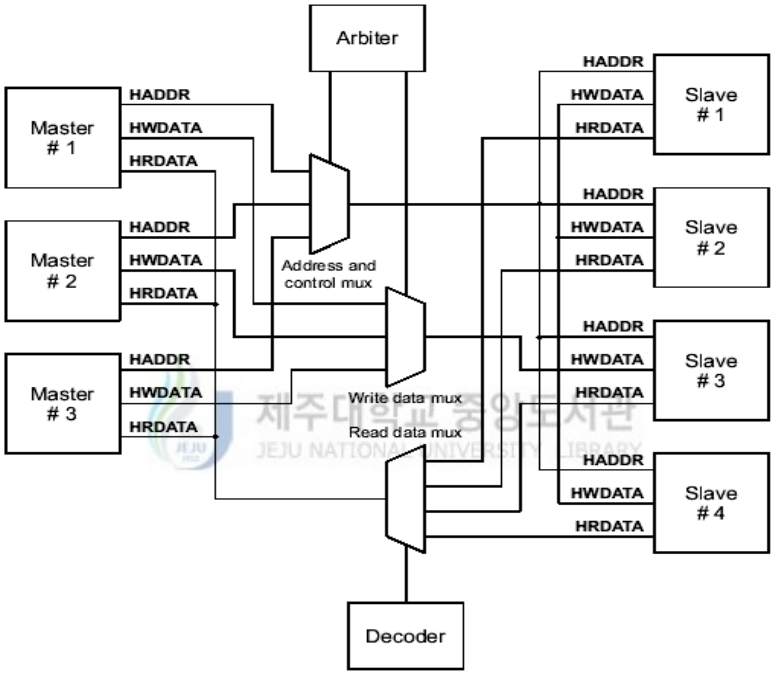

Fig. 21. Block diagram of SoC

(2) 해상도 변환기(resolution converter) 모듈 설계

해상도 변환기 모듈은 AMBA 버스를 통해 전달된 데이터를 가지고 레벨0, 레벨1, 레벨2의 탐색 영역 및 참조 영상을 만들어 내는 기능을 수행한다. AMBA 버스를 통해 전달된 32비트 탐색 영역 및 참조 영상은 2:1 샘플링을 통해 레벨1 데이터를 만들어내고 다시 2:1 샘플링을 하여 레벨2 영상을 만들어 낸다. Fig. 22.는 해상도 변환기를 Verilog HDL로 코딩하여 시뮬레이션한 결과이다. prev_resol_data[31:0]과 cur_resol_data[31:0]를 2:1, 4:1 샘플링하여 각각 16비트 및 8비트 데이터를 생성하는 것을 알 수 있다.

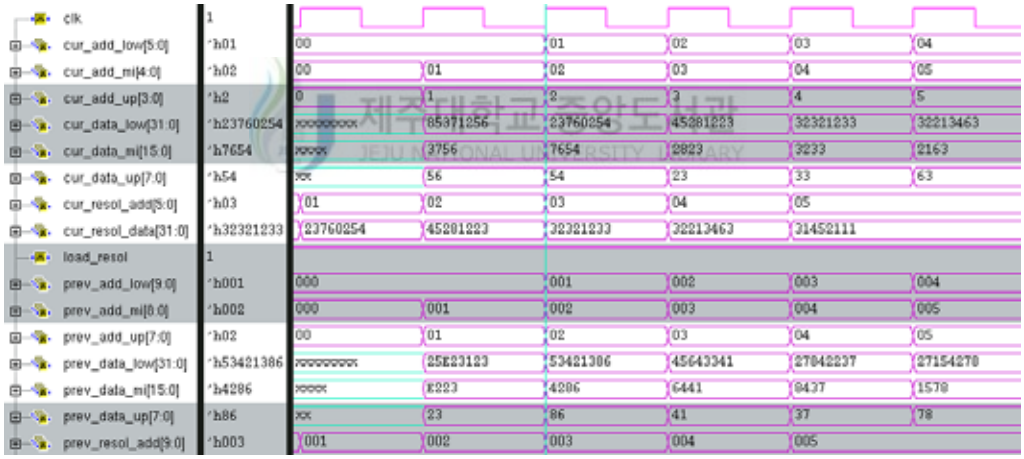

Fig. 22. Simulation result of resolution converter.

(3) 메모리 제어기(RAM Controller) 모듈 설계

메모리 제어기 모듈은 해상도 변환기에서 추출된 레벨2, 레벨1, 레벨0 영상 데이터를 RAM에 저장시키기 위한 제어 신호와 주소값, 데이터값을 발생시키는 모듈이다. Fig. 23.은 메모리 제어기 기능 블록도를 나타내는 그림이다. 그림에서 보는 바와 같이 5개의 하위블럭(sub-block)으로 나눠진다. DATA PATH unit은 RAM에 저장할 영상 데이터와 주소값, 제어신호를 RESOL_CNV에서 받아 SRAM_ARRAY에 전송하는 역할을 수행한다. PREV_ADD unit은 RAM에 쓰기가 완료된 후 RAM에 저장된 영상데이터를 읽어 오기 위한 제어 신호와 주소값을 발생시키는 역할을 한다. 마찬가지로 CUR_ADD unit은 RAM에 저장된 참조 영상을 읽어 오기 위한 제어 신호와 주소 값을 발생시킨다. RAM I/F unit 은 DATA PATH, PREV_ADD, CUR_ADD의 모든 제어 신호와 데이값을 수합하여 RAM ARRAY에 전달하는 역할을 수행한다. 이전 영상 및 참조 영상의 읽기 과정은 메모리 재사용 방법을 이용하여 고속으로 움직임 추정을 수행하도록 설계하였다. 메모리 재사용에 관한 자세한 내용은 3장에서 설명하였다.

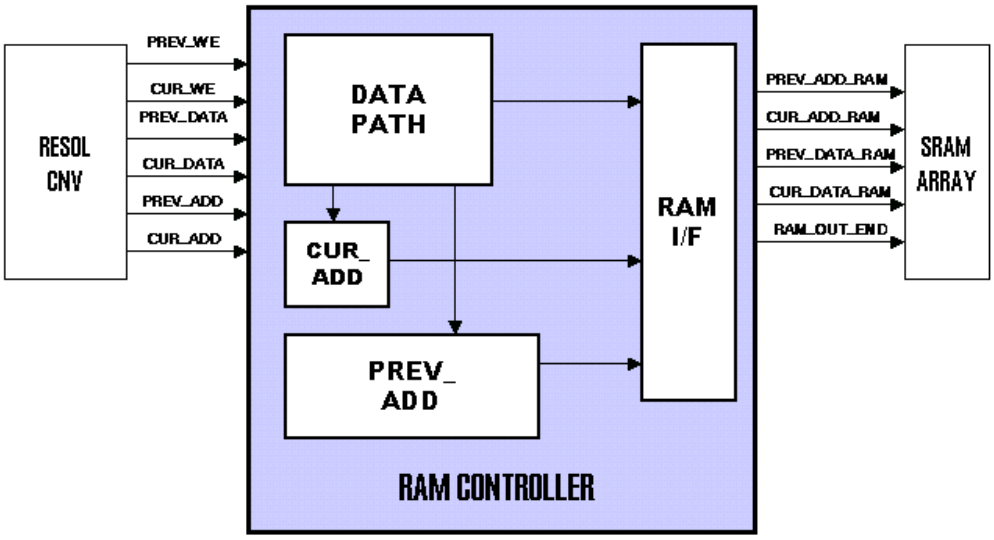

Fig. 23. Block diagram of ram controller.

Fig. 24은 메모리 재사용을 위한 데이터 이동 경로를 보여주는 그림이며 Fig. 25은 시뮬레이션한 결과이다. 첫 번째의 8x8 레지스터의 데이터 값은 inc_add 신호를 통하여 레지스터에 저장하게 되며 그 다음은 오른쪽 이동(right shift) 신호인 ri_add를 통하여 8x1 데이터가 새로이 들어오게된다. 3번의 오른쪽 이동이 끝나면 1x8데이터 크기의 다음 행 데이터가 들어오는데 아래 이동(down shift) 신호인 dw_add를 통하여 전달되며 끝으로 왼쪽 이동(left shift) 신호를 이용하여 8x1 데이터가 3번 들어오게 된다.

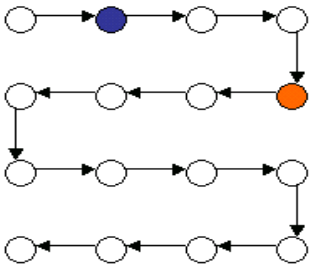

Fig. 24. Data flow

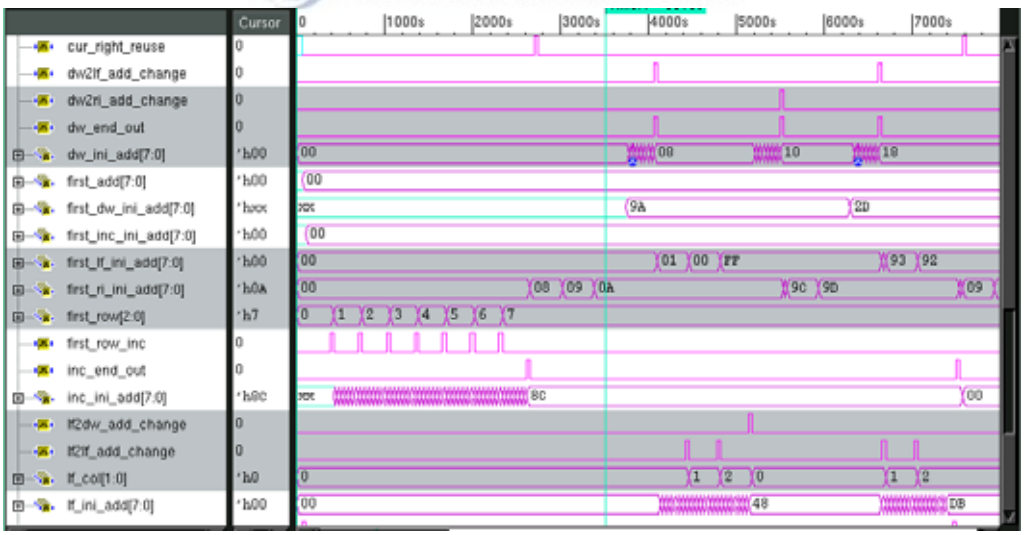

Fig. 25. Simulation result of ram controller

(4) 레지스터 배열기(REG_ARRAY) 모듈 설계

레지스터 배열기 모듈은 RAM으로부터 이전 영상 및 참조 영상을 받아 8x8 형식의 데이터로 만들어 PE_ARRAY로 전달하는 역할을 한다. 레지스터 배열기 모듈은 3개의 sub-block으로 나뉘는데 Fig. 26와 같이 크게 RAM_I/F 블록, PREV_REG, CUR_REG 블록이 있다. RAM_I/F 모듈은 RAM_ARRAY로부터 이전 영상 및 참조 영상을 받아 이전영상 레지스터 및 현재영상 레지스터에 데이터를 전달하는 역할을 한다. 이전영상 레지스터 모듈은 RAM_IF로부터 이전 영상 데이터를 받아 8x8 형식의 데이터로 만들어 PE_ARRAY로 전달하며 마찬가지로 현재영상 레지스터도 RAM_I/F로부터 참조 영상을 전달받아 PE_ARRAY로 전달한다. Fig. 27과 28은 이전영상 레지스터와 현재영상 레지스터를 Verilog HDL로 코딩한 후 시뮬레이션 한 결과를 나타낸다. Fig. 26에서 보는 바와 같이 이전 및 참조 영상을 입력 받은 후 8x8 형식의 prev_pe0 ~ prev_pe7, ref_pe0 ~ ref_pe7 데이터 신호를 내보낸다.

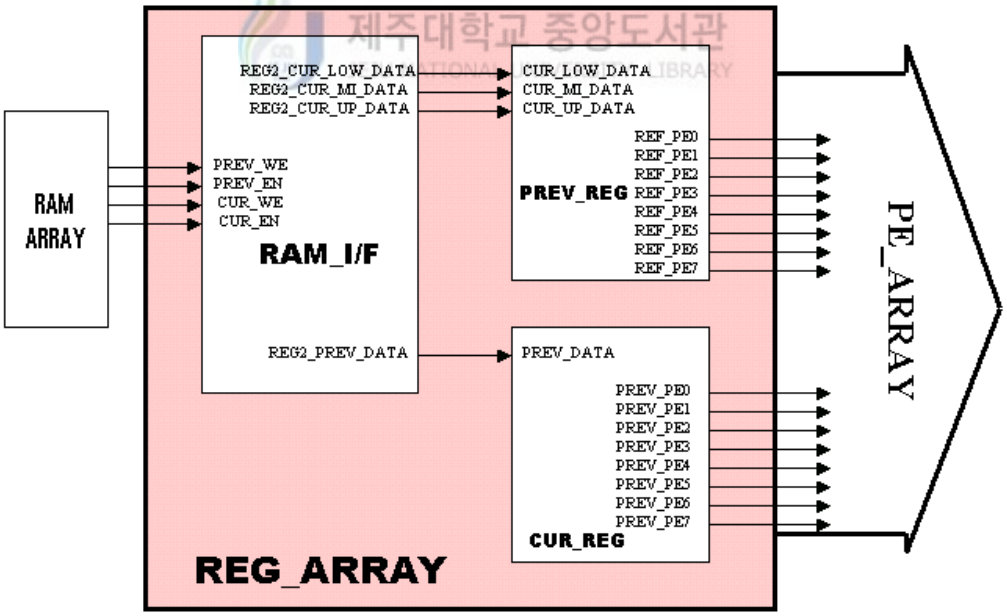

Fig. 26. Block diagram of REG_ARRAY

레지스터 배열기에서 가장 중요한 두 모듈은 이전영상 레지스터 배열기와 현재 영상 배열기인데 아래 Fig. 27.은 이전영상 레지스터 배열기를 Verilog로 기술하고 NC-Verilog 시뮬레이션툴을 이용하여 얻어진 결과이다. 이전영상을 8x8 레지스터로 수합하는 것이 주된 역할이며, 메모리 재사용 기법을 이용하므로 3장에서 설명한것과 같이 복잡한 레지스터 이동을 거쳐야만 한다. 증가상태(Increament)에서는 8x8 영상 데이터를 순차적으로 받아들이고 다음 상태(Right reuse)에서는 8x8 데이터를 왼쪽으로 쉬프트하고 남는 오른쪽 끝 8x1 데이터를 새로 받아들인다. 아래 시뮬레이션 결과는 여러 가지 상태에 따라 데이터를 왼쪽, 오른쪽으로 쉬프트하는 것을 보여주고 있으며 8x8 데이터를 이동하므로 64개의 출력 포트 중 일부분만을 보여주고 있다.

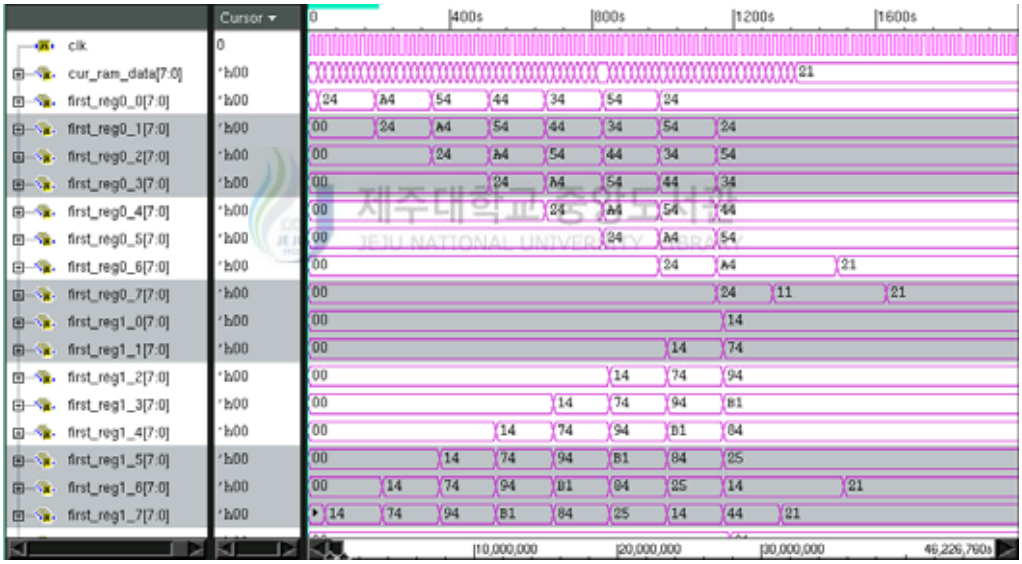

Fig. 27. Simulation result of PREV_REG

Fig. 28 그림은 현재영상 레지스터를 Verilog로 기술하고 시뮬레이션 한 결과를 보여주고 있다. 이전영상 레지스터와는 다르게 하나의 레벨에서 이용되는 현재 영상 데이터는 새롭게 받아들이는 기능없이 동일한 데이터를 이용함으로 레지스터 내의 이동이 이전영상 레지스터내의 데이터 이동보다 훨씬 적다.

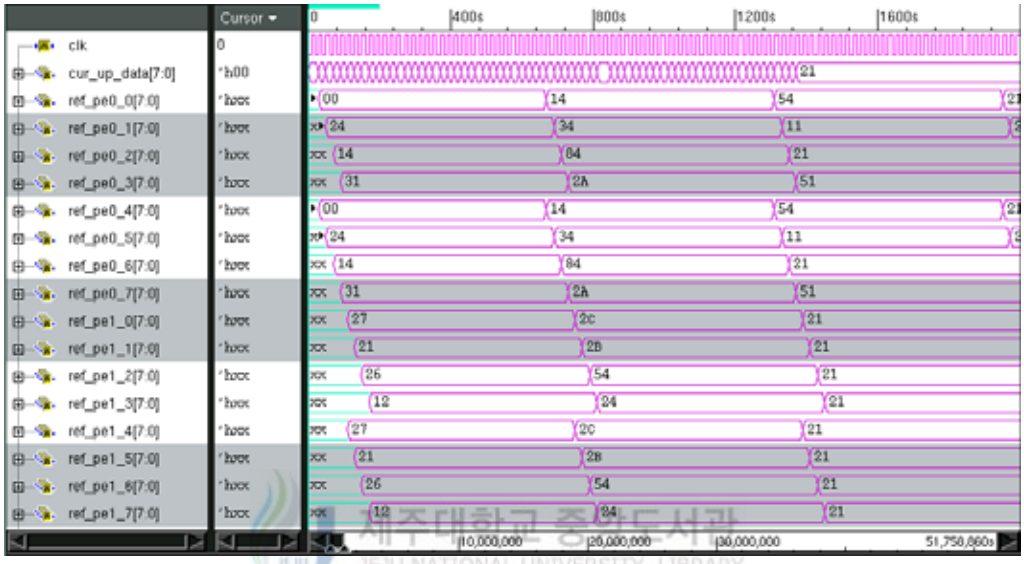

Fig. 28. Simulation result of CUR_REG

(5) PE 모듈 설계

PE(Processing Element) 모듈은 하나의 화소에 대해서 이전 영상 데이터와 현재 영상 데이터를 받아들여 차의 절대값을 구하는 기능을 한다. 그리고 PE 64가 모여 PE_Array를 만들게 된다. 그러므로 PE_Array는 8x8 블록단위로 차의 절대값을 구할 수 있다. Fig. 29.는 이전 영상 데이터와 현재 영상 데이터를 갖고 차의 절대값을 구하는 기능을 수행하는 것을 보여준다. C 프로그램으로 시뮬레이션한 결과와 일치함으로 차의 절대값을 수행하는 PE를 제대로 설계했음을 알 수 있다.

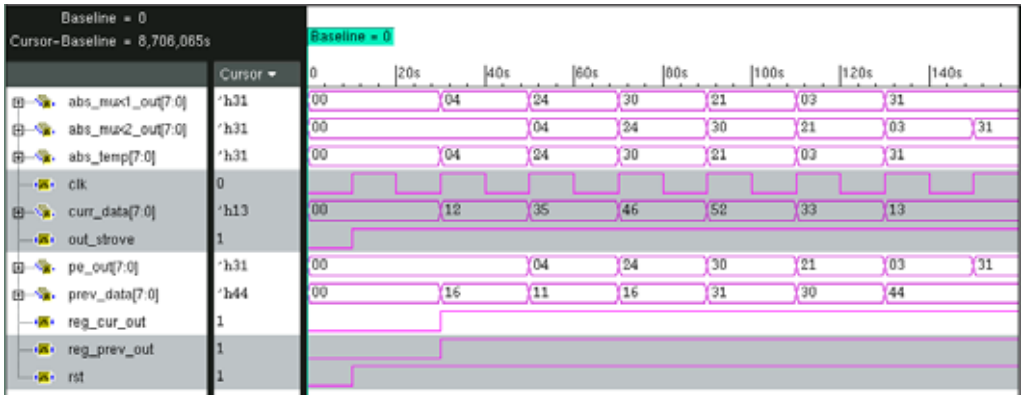

Fig. 29. Simulation result of PE.

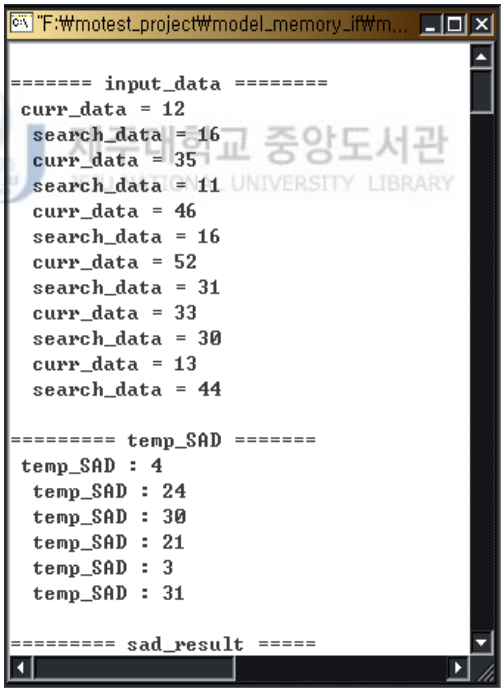

Fig. 30. C code program result of PE

(6) 비교단(Compare Unit) 모듈 설계

비교단 모듈은 PE_ARRAY로부터 계산된 SAD(이전 및 참조 영상의 차 절대값)을 바탕으로 최소가 되는 움직임 벡터를 찾는 모듈이다. Fig. 31는 비교단 모듈을 나타내는 기능 블록도이다. 그림에서 보는 바와 같이 세 개의 하위블럭(sub-block)으로 나눌 수 있는데 3개의 레벨에서의 최소 움직임 벡터를 찾는 기능을 한다. 비교는 먼저 레벨2에서 이루어지며 두개의 최소가 되는 움직임 벡터를 찾게된다. 여기서 구한 움직임 벡터는 레벨1에서의 움직임 벡터를 찾기 위해 다시 비교단에 간접적으로 들어오게 된다. 그 신호가 바로 UP1_MV, UP2_MV이다. 비교단의 시뮬레이션 결과는 다음 장의 FPGA 검증 부분과 중복되므로 다루지 않고 다음 장에서 자세히 설명한다.

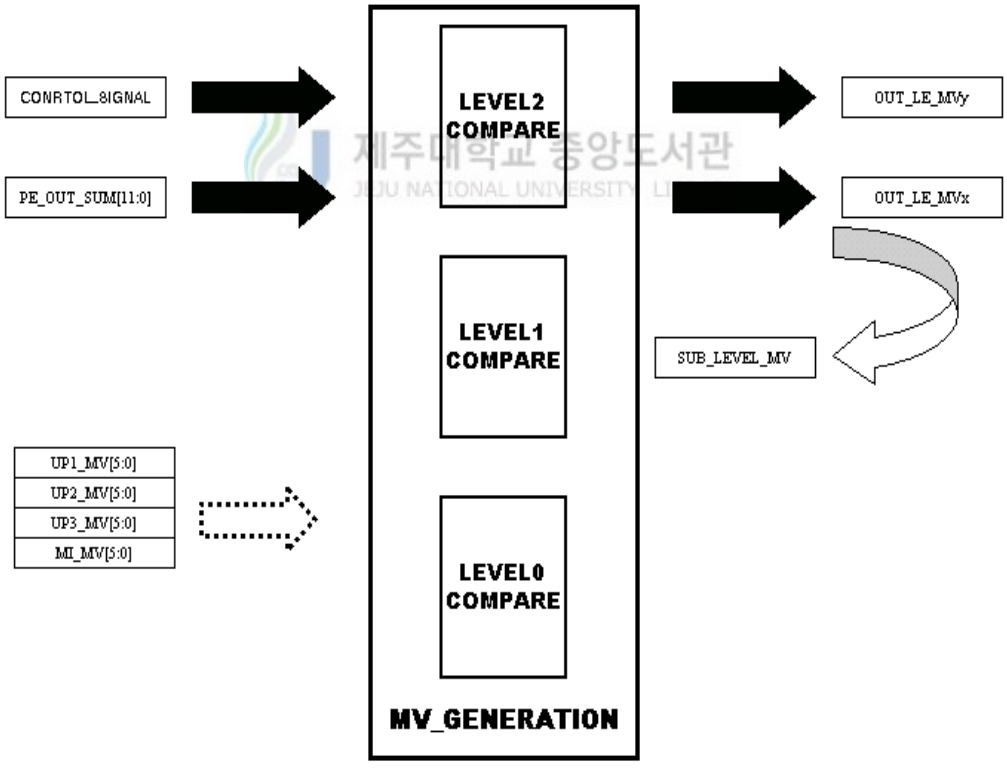

Fig. 31. Block diagram of Compare Unit.

(7) 전체 제어기(MRME_controller)

전체 제어기는 H.264 움직임 추정기를 총괄적으로 제어하는 모듈로 각 블록으로부터 상태 신호를 입력받고 현 상태에서 필요한 제어 신호를 넣어주는 기능을 한다. Fig. 32는 전체 제어기의 FSM(Finite State Machine)을 나타낸 그림으로 세 개의 레벨에서 움직임 벡터를 찾기 위해 Idle 상태부터 END 상태까지 단계적으로 필요한 제어신호를 출력하게 된다. Idle 상태에서 main_start 신호가 입력되면 해상도 변환(resolution convert)상태로 들어가서 세 개의 레벨로 탐색영역을 만들어 주고 RAM 배열에 데이터를 저장시킨다. 저장이 완료되면 RAM의 데이터를 읽어와서 레벨2(level2)에서부터 움직임 탐색을 수행하게 된다. 여러 단계를 거친 후 최종인 level0_3_process 상태에서 최적의 움직임 벡터를 추출하게 된다.

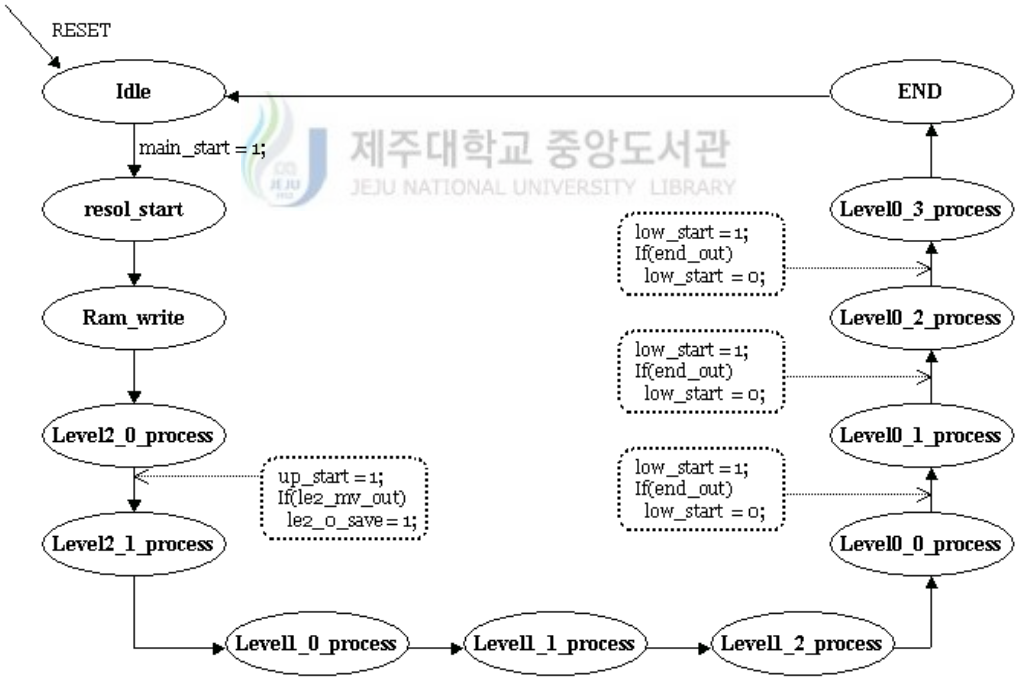

Fig. 32. FSM of main controller

(8) 움직임 추정기 전체 합성 결과

Fig 33은 H.264 움직임 추정기를 합성한 결과이다. 6개의 주요 블록이 있는데 제일 앞단의 모듈이 움직임 추정기 전체 제어를 담당하는 전체 제어기(mrme controller)이며 그 다음은 해상도 변환기(resol_cnv)이다. 그리고 메모리 제어기(ram_controller), 10개의 RAM으로 구성된 램 배열(ram_array), 다음이 레지스터 배열기(reg_array), 그리고 PE_ARRAY, 마지막으로 비교단(compare Unit) 모듈이다. 합성은 XILINX의 ISE 프로그램에 포함된 XST(xilinx synthesis tool)를 이용하여 합성하였다. 타켓 디바이스는 XILINX XCV600HQ240C(660KGates, 166개의 I/O)로 하였다. 10개의 RAM은 XILINX ISE 프로그램에 포함된 CORE GENERATION 툴을 사용하여 구현하였다.

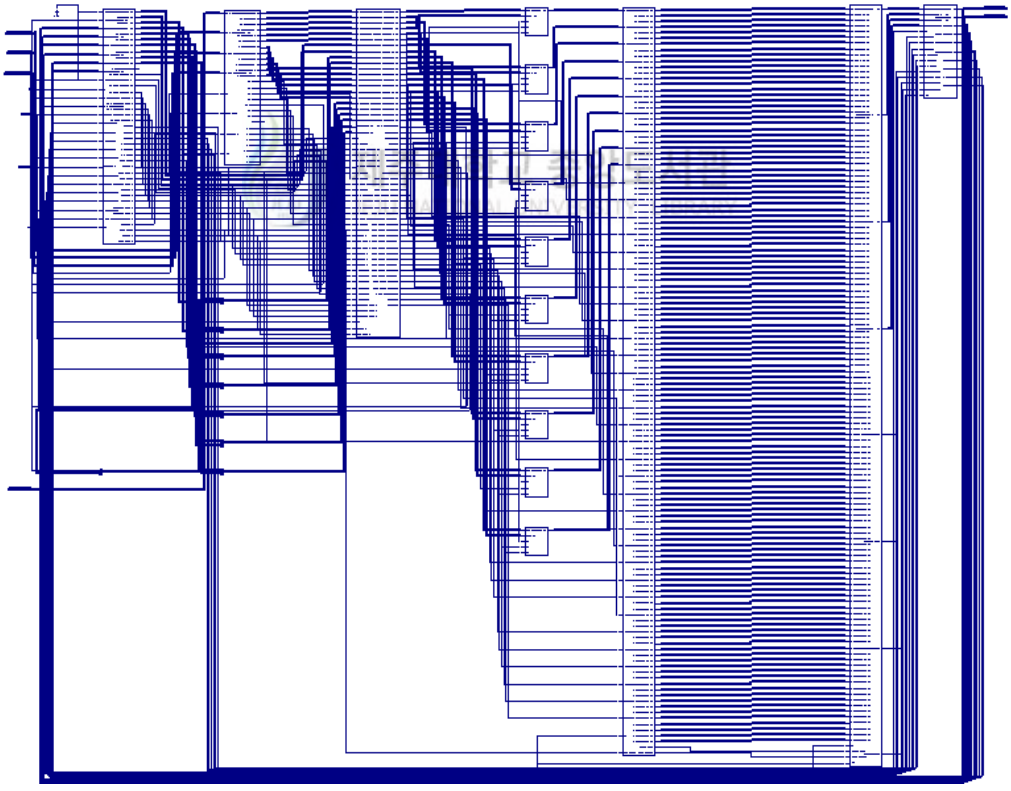

Fig. 33. Synthesis result of H.264 Motion Estimation.

2. 설계된 움직임 추정기 FPGA 기능 검증

(1) 검증용 영상 데이터 추출

1절에서 움직임 추정기의 주요 블록에 대한 설계와 검증이 이루어졌다면 2절은 설계된 움직임 추정기 전체 모듈에 대한 기능 검증을 수행한다. 검증에 사용된 영상은 CIF급 영상인 foreman을 이용하였다. Fig 34에 보는 바와 같이 왼쪽에는 이전 프레임 영상이며 오른쪽 프레임은 찾고자 하는 현재 프레임의 매크로블럭이 있다. 영상 데이터 추출은 Fig. 35 같이 간단한 C 프로그램을 이용하여 얻을 수 있고 이전 프레임의 탐색 영역 데이터와 현재 프레임의 매크로블럭을 추출하였다. 추출된 영상데이터는 FPGA 기능 검증용 입력 데이터로 이용하였다.

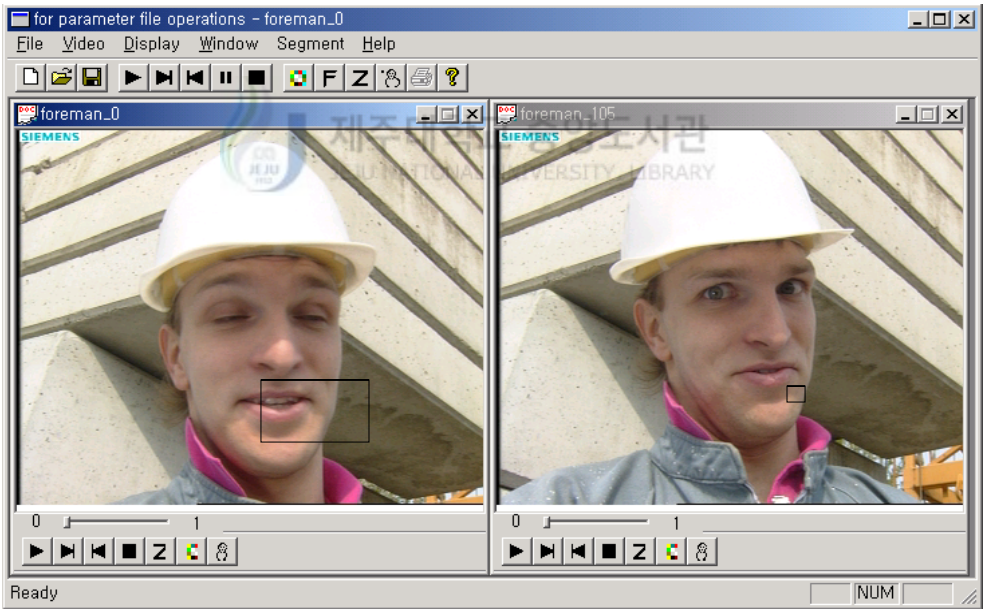

Fig. 34. Previous and current frame

```
#include <stdio.h>

void main()
{
        int i,j;
        unsigned char OrgImg[352];
        FILE *infile, *outfile;

        outfile = fopen("fore_image_6.txt", "wb");
        infile = fopen("foreman_0.yuv", "rb");
        if(infile==NULL){printf("FILE open error!!");return;}
        for(i=0;i<288;i++)
        {
            fread(OrgImg, sizeof(char),352, infile);
        for(j=0;j<352;j++)
                {
            fprintf(outfile, "%4d", OrgImg[j]);
                }
        fprintf(outfile, "\n");
        }
        fclose(infile);
        fclose(outfile);
}
}
```


Fig. 35. C code for drawing image

그림은 위에서 언급하였듯이 하나의 프레임에서 영상 데이터를 추출하기 위한 C 프로그램이다. CIF급(352x288) 영상 프레임에서 탐색 영역의 데이터 및 찾고자 하는 현재 매크로블럭의 데이터를 얻을 수 있다.

(2) 보드 레벨 검증 환경

설계된 H.264 움직임 추정기를 보드 레벨에서 검증하기 위해서 다음과 같이 검증 환경을 구성하였다. 움직임 추정기 IP(intellectual property)는 Xilinx FPGA에 하드웨어 프로그래밍하였고 움직임 추정기를 제어하고 영상 데이터를 전달하며 결과값을 읽어오는 기능은 Jupiter MCU가 담당하도록 하였다. 보드에서 사용한 MCU는 SE3208 코어를 내장한 (주)에이디칩스의 Jupiter이고, FPGA는 Xilinx사의 XCV600HQ240C이다. FPGA 하드웨어 프로그래밍은 Xilinx사에서 제공되는 ISE 6.1i 툴을 이용하여 움직임 추정기를 합성, 구현(implementation), 비트(bit)파일 다운로드하여 설계하였고 펌웨어 구현은 에이디칩스사에서 제공되는 ESIC Studio 툴을 이용하여 설계하였다. Fig. 36 그림은 FPGA와 MCU의 신호 연결을 나타낸 그림인데 데이터, 주소 값을 주고 받는 신호선과 제어를 위한 신호선이 서로 연결되어 있으며 Fig. 37는 Xilinx FPAG 보드와 Jupiter MCU 보드 사진이다. 검증 방법은 이전 영상 데이터와 현재 영상 데이터를 포함하고 MCU를 제어하기 위한 펌웨어를 시리얼 포트1(RS-232C)를 통하여 MCU에 전달하면 MCU는 탐색 영역의 이전 프레임 데이터와 찾고자 하는 현재 프레임의 매크로블럭 영상 데이터를 FPGA로 전달한다.

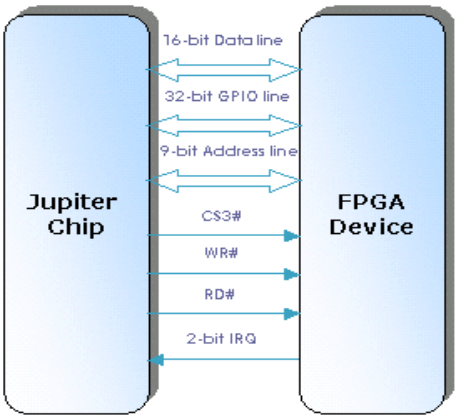

Fig. 36. Interface architecture

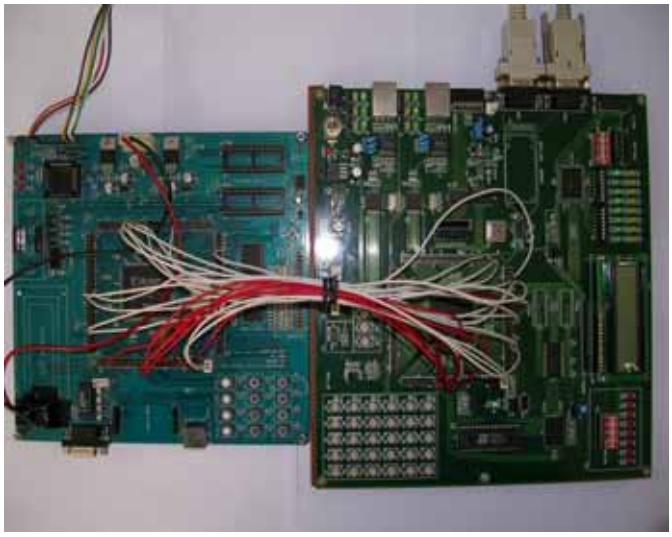

Fig. 37. Test board for motion estimation

움직임 추정기가 구현된 FPGA는 최적의 움직임 벡터를 추출하여 MCU로 전달하며 데이터값을 받는 즉시 시리얼 포트2(RS-232C)를 통하여 호스트 컴퓨터의 하이퍼터미널에 전달받은 결과값을 나타내 준다. Fig. 38은 움직임 추정기 검증 환경을 보여주는 그림이다.

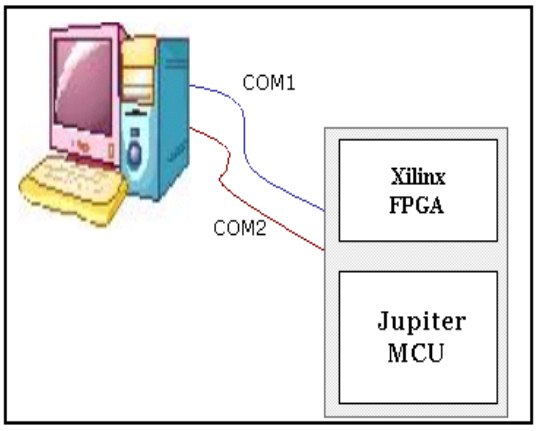

Fig. 38. Design verification environmemt

(2) FPGA 검증 결과

개선된 MRME 알고리즘을 FPGA에 구현하고 원하는 기능을 수행하는지 검증하였다. H.264 움직임 추정기는 세 개의 해상도 레벨에서 각각 움직임 벡터를 찾는다. Level2에서는 두 개의 움직임 벡터를 찾게되며 레벨1에서는 레벨2에서의 움직임 벡터 포인터를 중심으로 최소가 되는 움직임 벡터 하나를 구하게 된다. 마지막으로 레벨0에서는 레벨1의 움직임 벡터 포인터를 중심으로 최소가 되는 움직임 최종적으로 찾게 된다. Fig 40은 하이퍼터미널을 이용하여 레벨2 움직임 벡터값을 FPGA에서 읽어온 데이터를 나타낸다. 레벨2에서의 가장 작은 SAD를 갖는 움직임 벡터로 (6,5)와 (2,5)가 생성되었고 시뮬레이션 결과와 FPGA에서 얻어진 결과와 일치함을 확인할 수 있었다. 레벨2에서 얻어진 2개의 최소 움직임 벡터를 중심으로 레벨1에서 움직임 벡터를 추출한 결과 5(0101)과 7(0111)이 생성되었는데 앞의 두 비트 “01”은 레벨2에서의 3개의 탐색 후보 중에서 첫 번째 탐색 후보를 나타내고 마지막 두 비트는 각 x축과 y축의 탐색 결과를 나타낸다. 따라서 레벨2에서 추출된 움직임 벡터 (6,5)를 중심으로 하는 움직임 포인터에서 (1,3) 포인터가 가장 최소의 SAD값을 나타내었다. Fig. 42는 레벨1에서 움직임 벡터를 추출하는 시뮬레이션 결과이며 Fig. 43의 FPGA에서 읽어온 결과값과 일치함을 알 수 있다. Fig. 44과 Fig. 45은 레벨1에서의 움직임 벡터 포인터를 중심으로 최적의 움직임 벡터를 생성한 결과를 보여주는 그림이다. 결과값은 x축 “1”과 축 “1”이 생성되었다.

Fig. 39은 움직임 추정기의 결과 데이터인 움직임 벡터가 각 레벨에서 생성되고 최적의 움직임 벡터 포인터인 P3를 찾는 과정을 나타낸 그림이다. P1의 위치는 레벨1에서 [x축 y축]으로 [-4 +2]가 되며 P2의 위치는 P1을 중심으로 [+1 +3]이므로 레벨0에서 [-10 +8]이 된다. 최종적으로 P3의 위치는 P2를 중심으로 [+1 +1]이므로 레벨0에서 [-11 +8]에 위치하게 된다. 이 위치가 우리가 찾고자 하는 최적의 움직임 포인터가 된다. 탐색 영역에서 [-11 +8]에 위치한 매크로블럭이 현재 프레임에서 찾고자하는 매크로블럭과 가장 유사하다고 할 수 있다.

Fig. 41의 (b)는 찾고자 하는 매크로블럭 데이터 값이고 (a)은 움직임 추정기에

서 추출된 움직임 벡터를 바탕으로 생성한 매크로블럭이다. 두 매크로블럭이 완전히 일치하지는 않지만 거의 동일한 매크로블럭을 찾은 것을 확인 할 수 있다. 두 매크로블록간의 오차는 현재 매크로블럭과 추정된 매크로블럭을 빼줌으로 구하며 그 오차 값을 인코더(encoder)에서 전송하며 디코더(decoder) 측에서 오차값과 추정된 매크로블럭을 더함으로서 현재 매크로블록을 복원시킬 수 있다.

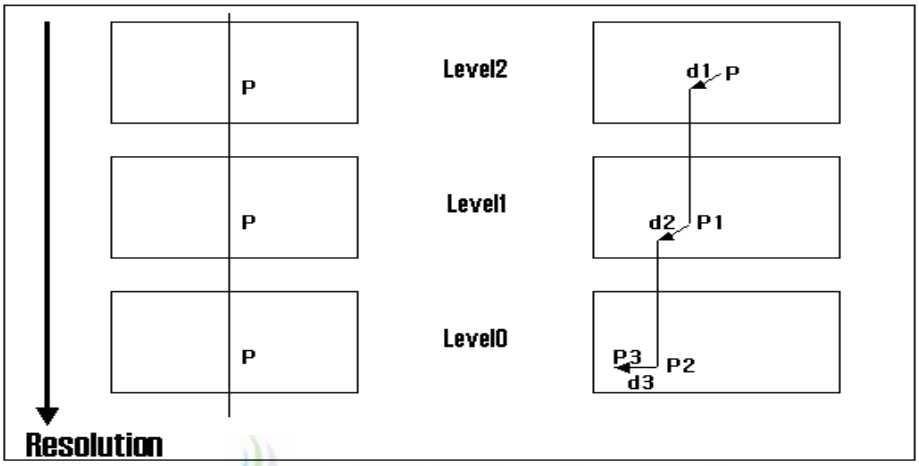

Fig. 39. hierachical motion vector search

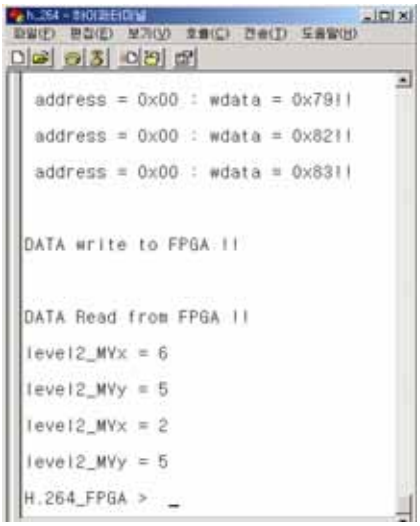

Fig. 40. Level2 simulation result values

236	236	235	235	235	235	236	236	235	236	235	236	235	235	235	234
235	236	236	236	234	235	235	235	235	236	236	236	235	235	235	234
236	235	235	236	236	236	236	236	236	236	236	235	235	235	235	236	235
236	236	236	236	236	236	236	236	237	235	234	235	236	235	234	235
235	235	235	235	235	234	235	236	236	236	235	235	236	236	234	235
235	236	235	235	235	235	235	235	235	236	236	236	235	236	235	235
235	235	235	235	235	235	235	235	236	235	235	235	235	235	234	235
236	236	236	236	236	235	236	236	235	235	235	236	236	236	236	235
236	236	235	237	236	236	236	236	236	237	237	235	236	236	235	235
235	235	237	236	235	236	235	235	235	236	236	235	236	236	236	235
236	236	235	235	235	236	236	235	236	236	236	234	235	236	236	235
235	236	236	235	235	236	235	235	235	234	235	236	235	235	236	237
235	235	236	236	235	236	235	235	236	235	235	236	235	235	235	235
235	236	237	237	236	236	235	235	236	236	235	236	236	235	235	236
235	236	237	235	234	235	236	234	232	234	235	235	235	235	236	236

(a). previous frame macroblock

236	237	237	237	237	237	237	237	238	237	236	238	236	236	227	214
236	235	236	237	236	236	237	237	236	237	237	237	237	235	236	228
235	235	235	235	236	235	235	237	236	235	235	237	236	235	235	235
236	236	235	236	236	235	235	236	237	236	236	236	236	236	235	235
236	236	234	235	236	236	235	236	236	234	234	234	236	236	235	234
236	236	234	235	236	236	234	234	235	235	236	235	235	235	234	234
237	235	235	236	235	235	235	235	235	235	234	234	235	235	235	236
236	235	236	235	235	235	234	235	235	235	235	234	234	235	235	235
235	234	235	236	236	236	236	236	235	235	235	235	235	236	235	235
235	235	235	236	236	235	235	235	236	235	234	234	235	235	235	235
235	235	235	235	236	235	235	236	236	235	234	235	235	235	235	235
236	235	236	235	235	236	235	234	235	235	234	235	236	235	235	236
237	236	235	235	235	236	236	234	235	236	235	234	234	235	235	235
236	236	235	235	235	234	235	235	235	235	235	234	234	235	235	234
235	235	235	234	234	235	235	234	235	236	235	234	234	235	235	235
235	235	235	236	235	235	235	235	236	235	233	234	234	234	235	235

(b). current frame macroblock

Fig. 41. 16x16 previous and current macroblock

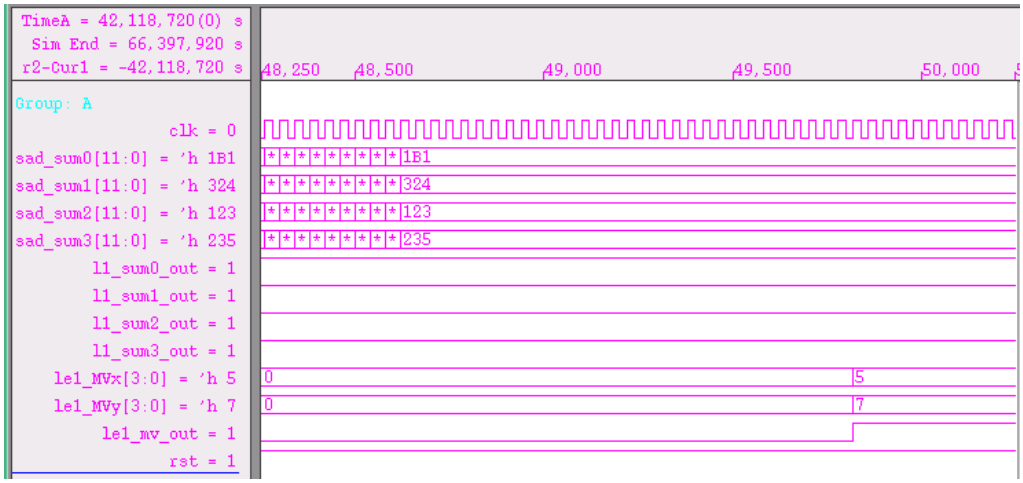

Fig. 42. Simulation result of Level1 compare

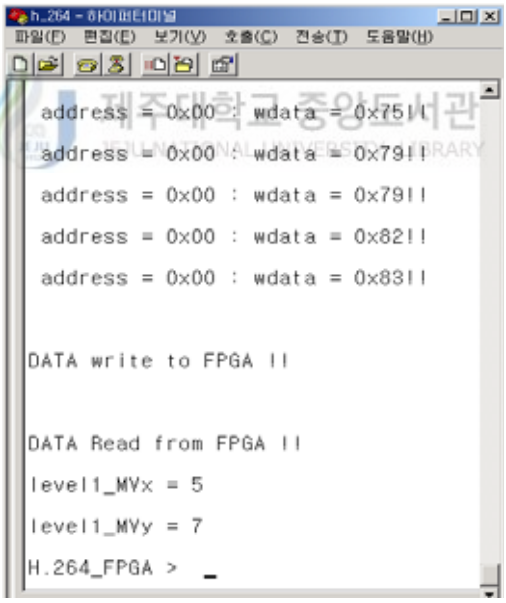

Fig. 43. Level1 simulation result value.

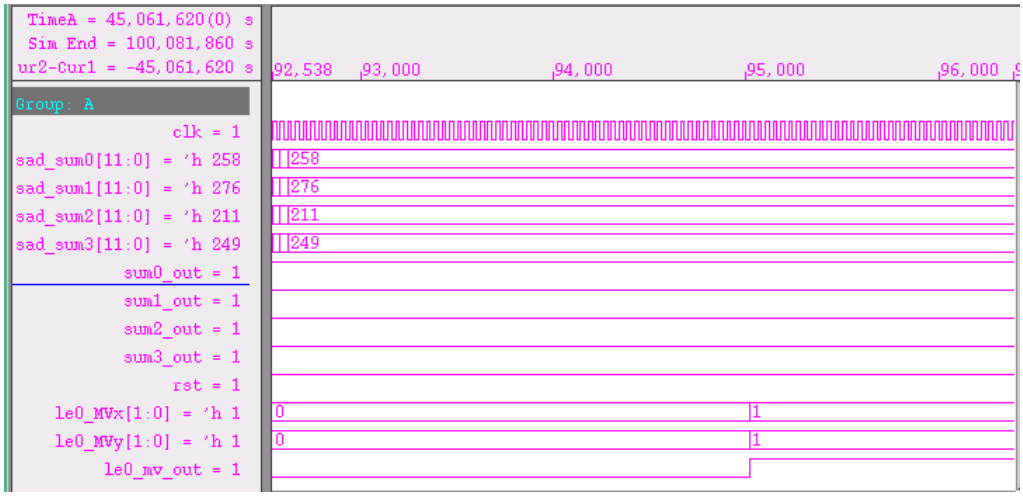

Fig. 44. Simulation result of Level0 compare.

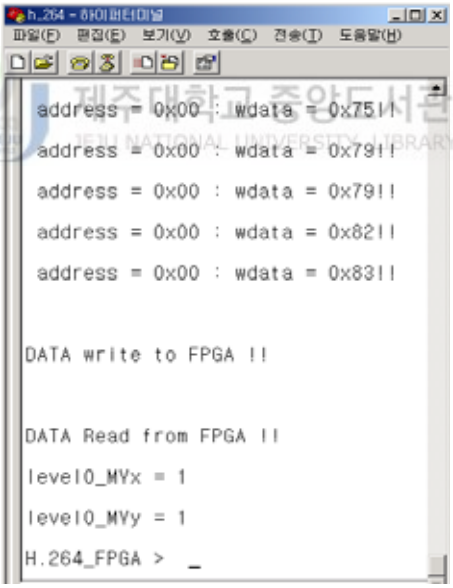

Fig. 45. Level0 simulation result value.

3 움직임 추정기 성능 평가

본 논문에서는 움직임 추정기에서 움직임 벡터를 효율적으로 찾아내기 위한 알고리즘 및 구조에 대한 기존 결과에 대해서 정리하고 본 논문에서 새롭게 제안된 MRME 알고리즘 및 구조에 대해서 정리하고 비교하였다.

제안된 알고리즘과 구조로 설계된 H.264 움직임 추정기는 CIF급(352x288), 30frames/sec 이미지를 적용하였을 때 데이터를 읽고오는 데이터 로딩(loading)에는 1656 clock이 소요되었고 데이터 처리(processing)에는 444clock이 필요하였다.

Fig. 48은 이것을 자세히 설명하여 주는 그림이다. Fig. 46은 초당 30프레임의 CIF 이미지를 처리하는 과정을 설명하는 그림인데 Vsync 일때 마다 하나의 프레임이 처리된다. 하나의 프레임에는 16x16 MB이 396개로 나눌 수 있으며 하나의 MB을 처리하는데에는 총 2100 Clock이 소요된다.

표1은 여러 논문에서 제안된 움직임 추정기의 PE 개수, 입력 속도, 처리 속도, 게이트 수등을 정리한 것이다. 이 표를 바탕으로 제안한 움직임 추정기의 성능을 비교하였다. Fig. 48에서는 하나의 MB을 처리하는데 필요한 클럭 수를 비교하였다. 효율적으로 설계된 움직임 추정기일수록 클럭수는 적다. 그림에서 알 수 있듯이 FS1이 클럭 수 1792로 가장 성능이 좋으며 제안된 움직임 추정기는 두 번째 성능인 2100을 필요로 하였다. Fig. 49은 하나의 CF(current frame)을 처리하는데 필요한 클럭수를 비교한 것이다. 마찬가지로 클럭 수가 적을수록 성능 향상된 움직임 추정기라 할 수 있다. Fig. 50는 하나의 MB을 처리하는 있어서 필요한 면적 크기와 클럭 수의 곱을 비교하였다. 면적과 클럭 수는 trade off 관계에 있는데 면적을 작게 만들면 클럭 수는 증가되어 성능이 저하된 움직임 추정기로 구현되며 클럭 수를 줄여 성능을 향상시키면 면적은 증가되는 현상이 발생하게 된다. 이동 단말기에 적용되는 영상 통신용 코덱에 들어가는 움직임 추정기라면 성능 감소를 감수하고서라도 면적을 줄여야하며 집 안에서 사용하는 영상 시스템일 경우에는 칩의 면적은 커지더라도 고화질을 대부분 요구하므로 성능 향상에 초점을 맞춰 설계해야 할 것이다. 본 논문에서 설계된 움직임 추정기는 하드웨어 크기가 114k로

다른 움직임 추정기에 비해 큰 게이트 수(gate count)를 나타나지만 성능은 다른 움직임 추정기에 비해 월등히 높다. 따라서 제안된 움직임 추정기는 경박단소화를 필요로 하는 이동 단말 보다는 집안 내의 디지털 TV나 홈네트워킹용 영상시스템에 적용하는 것이 유리할 것으로 판단된다. Fig. 51는 CIF급 이미지 처리의 기준이 되는 36.5MHz에서 처리할 수 있는 프레임 수를 비교한 것이다. 기준이 되는 36.5MHz는 초당 30프레임의 CIF급 이미지를 처리하는 필요한 대역폭으로서 30프레임을 처리하는 필요한 클럭수가 36.5MHz 가 넘으면 사실상 실시간 복원이 어렵게 된다. 그림에서 알 수 있듯이 제안된 움직임 추정기는 36.5MHz에서 43개의 프레임을 처리할 수 있어 초당 30 프레임을 넉넉히 처리할 수 있음을 알 수 있다.

Table. 1. Result of comparing hardware specifications

Architecture	MRMCS	ETRI	FS1	FS2	Proposed
Input data width	8	8	24	48	32
PE	5	16	256	64	64
Input Clock	3360	2100	768	384	1656
Processing Clock	2640	2400	1024	4096	444
Gate count	25K	29K	192.2K	N/A	114K

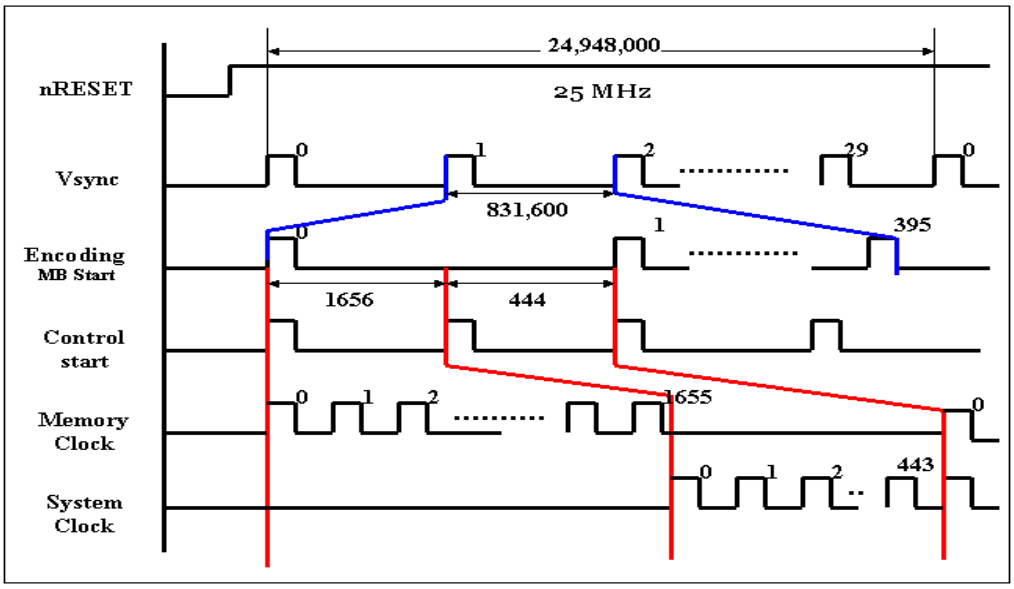

Fig. 46. Timing diagram of frame processing

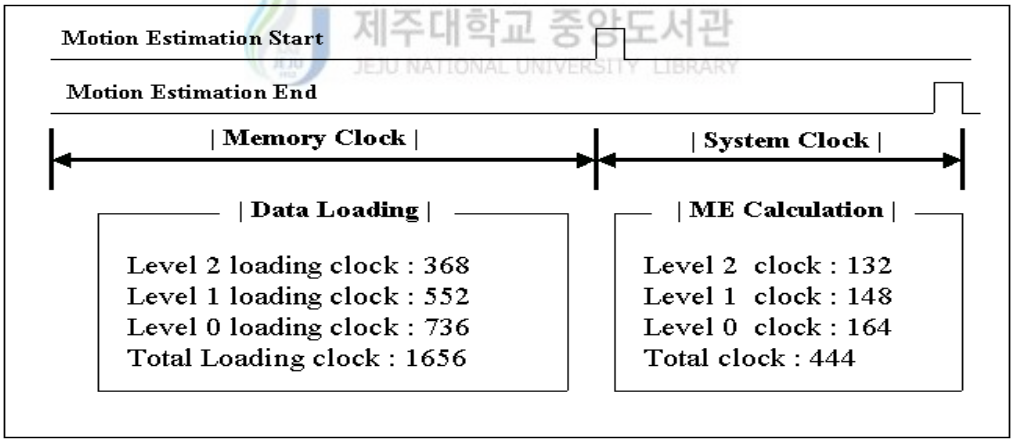

Fig. 47. Clock for motion estimation

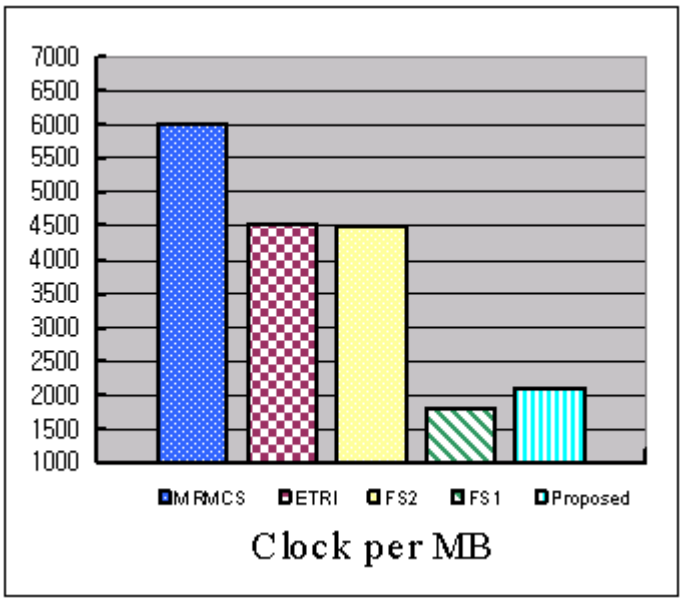

Fig. 48. Histogram of clock per MB

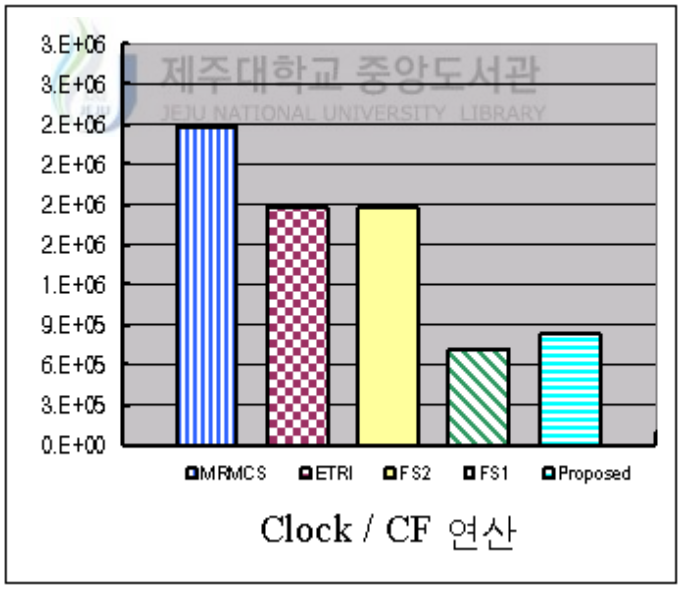

Fig. 49. Histogram of clock per CF

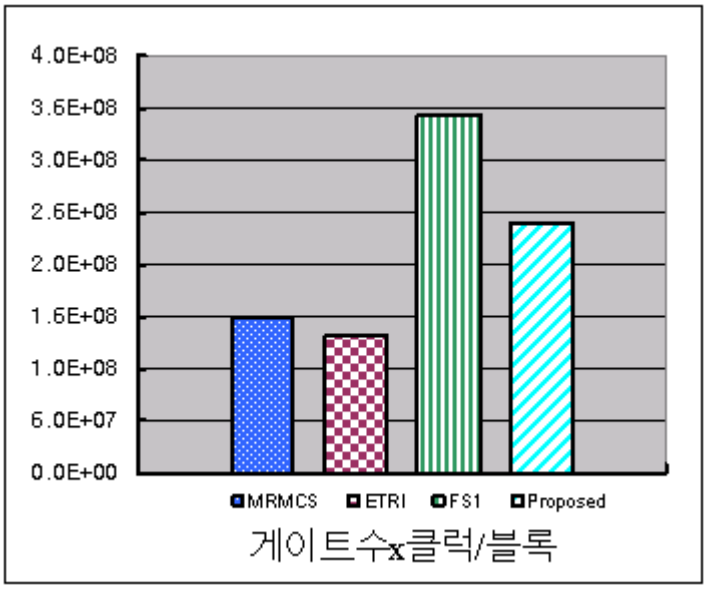

Fig. 50. Histogram of Area clock per BLOCK

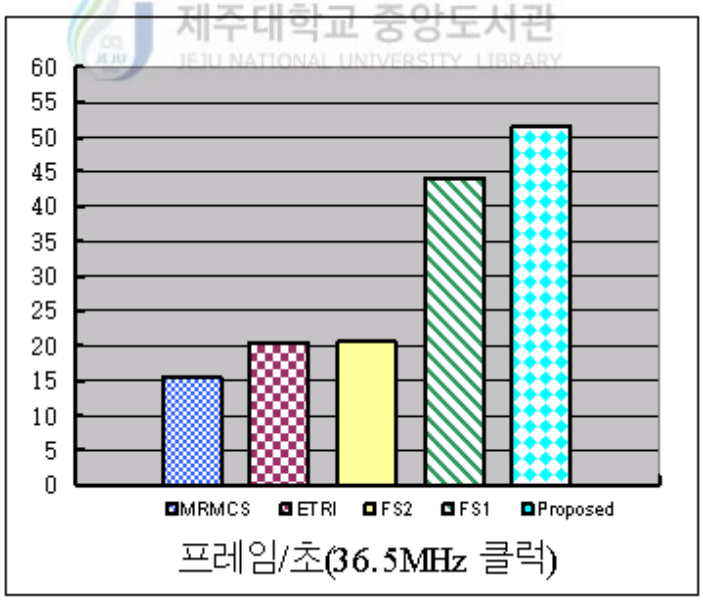

Fig. 51. Histogram of frames per second

Ⅴ. 결론

본 논문에서는 H.264 영상 시스템의 성능을 좌우하는 움직임 추정기를 설계하기위한 다양한 알고리즘를 분석하였고 이를 효율적인 하드웨어 구조를 제안하였다. 제안한 움직임 추정 구조를 Verilog HDL로 기술하고 FPGA상에서 구현하여 그 기능을 검증하였다. H.264를 위한 다해상도 움직임 추정 알고리즘의 구조를 기술하고 여러 기능을 추가하여 하드웨어로 제안된 모듈을 구현하였다. 이를 위해 다른 몇 가지의 움직임 추정 알고리즘과 제안된 구조의 성능을 비교하여 제안한 움직임 추정기가 동작 주파수, 처리 속도 면에서 우수한 성능을 나타내었다. 또한 MRMCS로 구현된 모듈과 비교하여 메모리 재사용을 위해 메모리 양이 추가되었으나 영상 데이터 입력속도 및 SAD 처리량에서 더 좋은 성능을 보이고 있다. 기본 탐색 유닛의 사용으로 4x4 서브 블록 단위로 정합 기준을 계산하고 이를 더 큰 블록의 정합 기준을 구하는데 이용함으로써 효율적인 하드웨어 구현이 가능하도록 하였다. 이렇게 구현된 전체 모듈은 다른 움직임 추정 알고리즘과 비교하여 낮은 동작 클럭으로 구현할 수 있었다. 제안된 움직임 추정기는 동작 주파수가 25MHz일때 CIF급(352x288), 초당 30 프레임 영상에 대해서 정화소 단위 움직임 추정을 처리할 수 있다. 또한 다해상도 움직임 추정기는 실시간으로 동영상 부호화가 가능한 성능을 가지고 있기 때문에 H.264 움직임 추정기를 하드웨어로 설계하기에 우수한 구조라고 할 것이다.

추후에는 H.264를 위한 부화소 단위 움직임 추정에 적합한 효율적인 하드웨어 구조에 대한 연구를 진행할 계획이다. 정화소 단위 움직임 추정 하드웨어와 부화소 단위 움직임 추정 하드웨어 구조가 효율적으로 결합될 수 있을 것이라고 예상된다. 또한 이렇게 구현된 칩은 압축 효율이 뛰어난 영상시스템에서 매우 유용하게 사용될 것이다.

참고 문헌

- A. Tamhankar and K. R. Rao, 2003. An Overview of H.264/MPEG-4 PART 10, 4th EURASIP conference.
- Ajay K. Luthra, Gary J. Sullian, 2003. Introduction to the Special Issue on the H.264/AVC Video Coding Standard, IEEE Trans. on Circuit and Systems for Video technology, vol. 13, No. 7.
- Cadence, 2002. NC-Verilog Tutorial for SimVision waveform viewer Users.
- C. H. Hseih, T. P. Lin, 1992. VLSI Architecture for Block-Matching Motion Estimation Algorithm, IEEE Trans. on Circuit and Systems for Video technology, vol. 2, No. 2.
- Chia-Wen Lin, Yao-Jen Chang and Yung-Chang Chen, 1998. Hierachical Motion Estimation Algorithm Based on Pyramidal Successive Elimination International Computer Symposium.
- H. Yeo, Y. Hen, 1995. A Novel Modular Systolic Array Architecture for Full-Search Block Matching Motion Estimation Algorithm, IEEE Trans. on Circuit and Systems for Video technology, vol. 5, No. 5.
- Jae Hun Lee, Kyoung Won Lim, Byung Cheol Song, and Jong Beom Ra, 2001. A Fast Multi-Resolution Block Matching Algorithm and its LSI Architecture for Low Bit-Rate Video Coding, IEEE Trans. on Circuit and Systems for Video technology, vol. 11.
- Junavit Chalidabhongse, C.-C. Jay Kuo, 2001. Fast Motion Vector Estimation Using Multiresolution-Spatio-Temporal Correlations, IEEE Trans. on Circuit and Systems for Video technology, vol. 11, No. 12.
- Lain E.G. Richardson, 2003. H.264 and MPEG-4 VIDEO COMPRESSION, WILEY. 281pp.
- Michael D. Ciletti, 2002. Advanced Digital Design with the Verilog

HDL,Prentice Hall. 979pp.

- Peter Kuhn, Algorithms, 2003. Complexity Analysis AND VLSI Architecture for MPEG-4 Motion Estimation, Kluwer Academic Publishers. 239pp.
- Seongmo Park, Miyoung Lee, Kwangki Ryoo, Hanjin Cho, Jongdae Kim, 1998. A MPEG-4 video codec chip with low-power scheme for mobile application, ETRI
- Viet L. Do, Kenneth Y. Yun, A Low-Power VLSI Architecture for Full-Search Block-Matching Motion Estimation.
- 김혁, 2003. Real XILINX FPGA World 엔트미디어, 2003. 436pp.
- 강성호, 김대정, 이승준, 이찬호,2004. HDL을 이용한 SoC 및 IP 설계기법, 홍릉과학출판, 271pp..
- 서울대학교 SoC설계기술사업단, 2004. 플랫폼기반 SoC 설계 교육 강의록.
- 인터에프씨, 2003. 순수국산 EISC 마이크로프로세서 JUPITER Chip 100% 활용 하기. 265pp.

감사의 글

반도체 설계를 공부하게 된다는 점과 새로운 사람들과의 생활에 대한 설레임으로 시작된 대학원 생활이 어느덧 2년이 지나 졸업을 앞두게 되었습니다. 지난 2년 동안 하고 싶었던 공부를 맘껏 할 수 있어서 행복했고 좋은 사람들과 함께 할 수 있어서 저에게는 너무나 소중한 시간이었습니다.

통신공학에 대한 남다른 열정으로 20대 청년 못지 않게 열심히 살아가시는 임재윤 교수님께 먼저 감사의 말씀을 드립니다. 대학원 생활동안 교수님의 부지런함과 패기를 보면서 많이 느끼고 배우게 되었습니다. 앞으로 제가 사회생활을 하는데 있어 커다란 밑거름이 될 것이라 확신합니다. 그리고 너무나 따뜻하게 저의 부족한 논문을 지도해 주신 이용학 교수님과 양두영 교수님께 감사드립니다. 언제나 인자하시고 부드러운 미소로 대해 주시던 문건 교수님과 김흥수 교수님께 감사드립니다. 학부때부터 많은 가르침을 주셨던 강진식 교수님 감사합니다. 그리고 첨단의 이동통신에 대해 모든 것을 가르쳐 주시려던 좌정우 교수님께 감사드립니다. 통신공학과의 모든 교수님 항상 건강하십시오.

재밌는 농담으로 즐겁게 해주시고 항상 밝고 따뜻했던 부식 선배님, 성욱 선배님, 권익 선배님, 봉수 선배님 정말 고마웠습니다. 친절하고 성격 좋은 이진호, 양윤희 조교 선생께도 감사의 마음을 드립니다. 대학원 선배로서 따뜻하게 대해 주었고 도움을 많이 주신 창윤이형, 영애 누나에게 감사의 마음을 전합니다. 그리고 대학원 동기인 진아에게 진실로 고마운 마음을 전합니다. 함께 연구실에서 공부할 수 있어서 즐거웠고 어려울 때마다 의지가 되어 대학원 생활을 잘 해낼 수 있었습니다. 반도체 설계에 대해서 나의 사수가 되어준 현미에게도 감사의 마음을 전합니다. 그리고 열심히 일하고 또 열심히 잘 뭉치는 모습이 보기 좋은 정현이형, 재오, 성민, 인경, 영주에게도 고마움을 전합니다. 회사가 반드시 번창하길 기원합니다. 그리고 학교는 다르지만 같이 대학원에 입학해서 열심히 공부하는 영호형, 민석에게도 감사를 드리고 반드시 꿈을 이루길 바랍니다. 마지막으로 대학원에 입학하게 도와주신 부모님 사랑합니다.