論文98-35S-9-11

변형된 4스텝 써치를 이용한 블럭정합 움직임 추정 및 보상 알고리즘의 VLSI 구조 설계

(VLSI Architecture Designs of the Block-Matching Motion Estimation/Compensation using a Modified 4-Step Search Algorithm)

李 東 澔 *

(Dong-Ho Lee)

요 약

본 논문에서는 기존 고속 블럭 정합 알고리즘보다 성능이 우수하고 하드웨어 구현에 적합한 새로운MFSS (Modified Four-Step Search) 알고리즘을 제안한다. 제안하는 알고리즘의 추정 과정은 일정한 규칙을 갖기 때문에 하드웨어 구현에 적합하고, 모의실험을 통해 거의 FS(Full Search) 성능에 근접할 정도로 기존의 고속 움직임 추정 알고리즘보다 성능의 우수함을 확인하였다. 본 논문에서는 이러한 MFSS(Modified Four-Step Search) 움직임 추정 알고리즘을 이용한 효율적인 움직임 추정 및 보상기의 VLSI 구조를 제안하고 설계 결과를 소개한다. 움직임 추정 및 보상기 설계에서 중요한 고려 사항은 설계 결과의 하드웨어적인 크기와 출력이 나오기 까지의 필요한 지연 시간인데, 본 논문에서는 9개의 PE(Process Element)만를 이용하여 구현함으로써 전체 로직의 양을 최적화 하였고, 움직임 추정기와 보상기를 결합함으로써 메모리를 공유하고 필요한 지연시간도 줄이는 구조를 제안한다.

Abstract

This paper proposes a new fast block-matching algorithm, named MFSS(Modified Four-Step Search) algorithm, which has better performance and is more adequate for hardware realization than the existing fast algorithms. The proposed algorithm is suitable for hadware realization since it has a unique regularity during the search procedure. It is shown from simulation results that its performance is close to that of FS(Full Search) algorithm. This paper also proposes a VLSI architecture and presents some design results of a motion estimator and compensator which adopted the MFSS algorithm. The important aspects considered in designing a motion estimator and compensator are hardware complexity of design results, and total delay needed to generate the motion compensated data after finding the motion vectors. Hardware complexity is minimized by using just nine PE(Process Element)'s, and total delay is minimized by sharing search memory of the motion estimator and compensator.

Ⅰ. 서 론

최근의 동영상 압축기술은 멀티미디어 통신, 비디오 폰, 원거리 화상회의, DVD, HDTV등 그 응용 분야가 점차 확대되고 있다. 영상 압축 기술의 핵심은 연속되는 비디오 신호에 존재하는 시간적, 공간적 정보

* 正會員, 漢陽大學校 制御計測工學科
(Dept. of Control & Instru. Eng., Hanyang Univ.)

接受日字: 1998年3月4日, 수정완료일: 1998年7月4日

의 중복성 이용에 있다. 시간적 정보의 중복성을 제거하는 방법으로는 동영상 신호의 움직임 보상 부호화 기법이 적용된다. 특히 움직임을 추정하는 부분은 방대한 연산을 필요로하기 때문에 동영상 부호화기의 실시간 구현에 어려움이 따른다.

움직임 벡터 추정 방법으로는 블럭단위로 움직임 정보의 유사성을 계산하는 블럭 정합 알고리즘(BMA: Block Matching Algorithm)이 많이 이용된다. 블럭 정합 알고리즘에는 FS(Full-Search) 알고리즘을 비롯하여 TSS(Three-Step Search) 알고리즘[2], HS(Hierarchical Search) 알고리즘[3], FSS(Four-Step Search) 알고리즘[4]등의 많은 추정 알고리즘들이 발표되었다. 이 중 FS는 가장 좋은 성능을 갖지만 많은 연산량이 필요하므로 실시간 구현에 어려움이 있고, HS는 고속 알고리즘의 가장 이상적인 형태이나 역시 연산량이 많고 구현을 위한 VLSI 구조가 복잡하다. TSS는 가장 많이 사용되는 대표적인 고속 탐색 알고리즘이지만 통계적인 움직임 벡터를 고려하지 않고 모든 움직임 벡터의 가능성을 같게 보기 때문에 실제 적용에서는 비효율적인 면을 나타낸다. 그 반면 FSS는 TSS와 비슷한 양의 연산으로 TSS 보다 높은 성능을 갖는다.

블럭정합 움직임 추정 알고리즘의 하드웨어 구현에 관해서도 많은 연구 결과가 발표되었다. Array Processor를 이용한 구현 방법[11][12], FS(Full Search) 알고리즘[13][14], HS(Heuristic Search) 알고리즘[15], TSS(Three Step Search)[16]등 특정한 알고리즘에 근거한 구현 방법 및 구조에 관한 연구가 수행되었다. 앞에서 설명하였듯이 FS와 HS 알고리즘은 우수한 성능을 갖지만 많은 연산량이 필요하기 때문에 실시간 구현에 어려움이 따른다. 고속 탐색 알고리즘 중에서 가장 자주 사용되는 TSS의 경우에는 연산량은 적지만 성능이 우수하지 못하고, 구현상에서도 알고리즘이 단계별로 탐색점의 배치가 틀리기 때문에 하드웨어 구조가 복잡해진다.

본 논문에서는 FSS 알고리즘을 변형하여 FS보다 적은 수의 연산으로도 FS에 거의 근접하는 성능을 갖고 하드웨어 구현에 적합한 MFSS(Modified Four-Step Search)알고리즘을 제안한다. 또한 제안하는 MFSS 알고리즘에 근거한 효율적인 움직임 추정 및 보상기의 구조와 VHDL을 이용하여 설계된 결과를 소개한다. 특히 9개만의 병렬 연산 요소를 이용한 구조를 제안하고, 움직임 추정 및 보상을 위해 필요한 메모리의 효율적인 구조를 제안한다. 움직임 추정 및 보상기 설계에서 추정 성능이외에 중요한 고려 사항은 설계 결과의 하드웨어적인 크기와 출력이 나오기까지의 지연 시간이다. 본 논문에서는 움직임 추정기와 보상기를 결합함으로써 메모리를 공유하고 필요한 지연시간도 줄일수 있도록 설계하였다. 또한 필요한 지연시간을 단축시키기 위해 데이터의 통신 버스를 32 비트로 확장하여 4개의 데이터를 병렬로 처리하였다.

본 논문의 주요 구성은 다음과 같다. II 장에서는 새로운 MFSS 추정 알고리즘을 제안하고 모의실험을 통한 성능 결과를 소개한다. III장에서는 MFSS알고리즘을 근거로 하는 움직임 추정기의 구조에 관하여 설명하고, IV장에서는 움직임 보상기의 구조 및 설계에 관하여 설명한다. V장에서는 VHDL을 이용하여 설계한 움직임 추정기와 보상기의 설계 결과를 소개한다. 마지막으로 VI장에서 결론을 맺는다.

II. Block-Matching Algorithm

1. MFSS(Modified Four-Step Search) 알고리즘

FS 알고리즘은 탐색 영역 내의 모든 탐색점과 비교하는 방법이다. 이 알고리즘은 모든 탐색점에 대해 비교하기 때문에 탐색 알고리즘 중 가장 뛰어난 성능을 갖지만 연산량이 많아 실시간의 하드웨어 구현에 어려움이 있다.

TSS(Three-Step Search) 알고리즘은 고속 알고리즘 중 가장 널리 알려진 알고리즘으로 FS 알고리즘에 비해 계산량을 크게 줄였으며 움직임이 적은 영상에 대해서는 적응력이 떨어지지만 움직임이 큰 영상에 대해서는 어느 정도 신뢰할 만한 움직임 벡터를 구할 수 있는 알고리즘이다. TSS 알고리즘이 움직임 벡터의 통계학적 분포를 고려하지 않고 모든 움직임 벡터의 확률이 동일하다고 보고 탐색점을 선정하기 때문에 성능이 떨어지는 문제점이 있다. 이를 보완하기 위해 움직임 벡터의 통계학적 발생 확률을 고려하여 탐색점을 선정한 FSS(Four-Step Search) 알고리즘이 소개되었다.[5] 통계학적으로 보면 움직임 벡터의 분포가 대부분 -2 ~ 2 사이에 분포한다는 것을 알 수 있다. 따라서 이 알고리즘에서는 첫번째 단계에서 -2 ~ 2 사이의 9 점을 탐색점으로 선정하여 탐색을 시작하였다. 움직임이 적은 영상에서는 TSS 보다 정확한 움

직임 벡터를 구하지만 움직임이 큰 영상에 대해서는 성능이 떨어지는 단점이 있다.

본 논문에서 제안하는 MFSS 알고리즘은 기존의 FSS의 구조를 변형하여 FS보다 적은 수의 연산량으로 거의 FS에 근접하는 추정 성능을 갖는 탐색 알고리즘이다. 그림 1에는 MFSS 알고리즘을 이용하여 움직임을 추정하는 예를 도시적으로 나타내었다. 기존의 TSS나 FSS 알고리즘은 움직임이 크거나 작은 영상에 대해 각기 장단점이 있고 탐색점의 간격이 매 스텝마다 달라 하드웨어 구현 시 복잡해지는 단점을 갖는다. 이에 반해 제안하는 MFSS 알고리즘은 그림 1에서와 같이 각 스텝은 탐색점의 간격이 일정하고, 9개의 탐색점으로 구성되는 기본 모듈로 구성되기 때문에, 하드웨어 구현이 용이하고 움직임 추정에 대해서도 충분한 탐색점의 분포로 움직임이 많고 적음에 크게 영향을 받지 않는다. MFSS알고리즘의 구체적인 탐색 방법을 설명하면 다음과 같다

Step 1. 원점과 주위 8개의 탐색점을 포함하여 9점 중에서 최소 에러점을 찾는다.

Step 2. 이전 스텝의 최소 에러점을 기준으로 36점 중에서 최소 에러점을 구한다. 이 때 탐색점들의 분포는 기준점에서 좌측상단 쪽으로 3, 우측 하단으로 2의 범위내의 점들이 된다.

Step 3. 이전 스텝의 최소 에러점을 기준으로 36점에 대해 최소 에러점을 구한다. 이 때 탐색점의 분포는 분포의 중심에 있는 4점 중에 탐색점의 분포가 원점에서 멀어지는 방향으로 설정한다.

Step 4. 이전 스텝의 최소 에러점을 기준으로 9점에 대해 최소 에러점을 찾는다. 이 때 찾아진 점과 원점과의 거리가 움직임 벡터가 된다.

Step 1에서 원점을 중심으로 9점만을 탐색하는 이유는 먼저 개략적인 움직임의 방향을 찾은 다음 Step 2에서 Step1의 탐색영역을 포함하는 넓은 영역을 탐색하기 때문에 성능이 우수해진 다는 것을 실험을 통해 입증하였다. <표 1>에는 하드웨어 구현시 복잡도를 나타내는 각 알고리즘들의 최대 탐색점수를 비교하였다. 표를 보면 MFSS 알고리즘이 TSS나 FSS보다는 탐색점 수가 다소 증가하지만 성능면에서는 거의 FS에 가깝다는 사실을 알수 있다.

그림 1에서와 같이 MFSS 알고리즘의 모든 스텝은 9 점으로 구성되는 기본 모듈로 구성되어 하드웨어로 구현하기에 적합한 구조를 갖는다. 첫번째 스텝과 네번째 스텝은 1개의 기본 모듈로 구성되고 나머지 스텝들은 4개로 구성된다. 각 기본 모듈은 9점으로 이루어져 있는데 이는 다음 장에서 소개하는 하드웨어 구조에서 에러를 계산하는 PE(Process Element)가 9개가 되는 이유이다. 9개의 PE를 이용하여 병렬로 구현할 경우, 각 알고리즘의 첫번째 스텝이 완료되기 위해서는 TSS의 경우에는 576 사이클, FSS의 경우에는 400 사이클, MFSS의 경우에는 324 사이클 이상이 필요하게 된다.

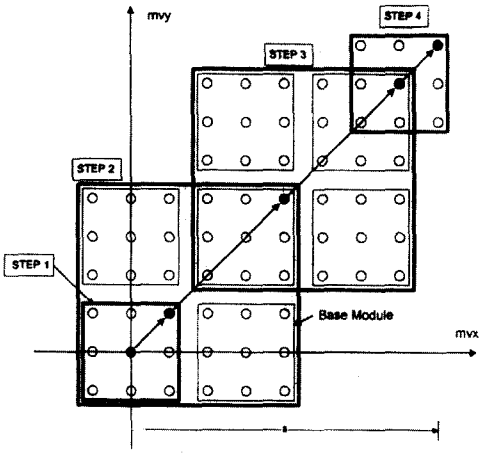


그림 1. MFSS 알고리즘의 예
Fig. 1. An search example of the MFSS algorithm.

표 1. 각 탐색 알고리즘에 대한 최대 탐색점수
Table 1. Number of maximum search points for search algorithms.

탐색방법	FS	TSS	FSS	MFSS
탐색점수	256	27	36	81

2. 모의실험 결과

그림 2에는 352x240 크기의 'Flower' 영상에 대해 FS, TSS, FSS, MFSS 알고리즘을 적용하여 움직임 보상된 영상의 PSNR을 나타내었다. 'Flower' 영상은 화면이 전체적으로 일정한 속도로 빠르게 움직이는 영상이고, 화면에 고주파 성분이 많아 움직임 추정 알고리즘들의 전체적인 성능은 전체적으로 낮다. 움직임 추정 성능은 MFSS 알고리즘이 다른 고속 알고리즘 보다는 성능이 우수하고 거의 FS 성능에 근접

하다는 것을 알수있다. <표 2>에는 'Flower' 영상과 같은 크기의 'Football', 'Bicycle', 'Mobile' 영상의 모의실험 결과의 평균 PSNR을 나타내었다. 평균 PSNR을 비교하였을때, MFSS 알고리즘의 움직임 추정성능이 기존의 고속 움직임 추정 알고리즘에 비해 0.5 dB 이상 높은 것을 알수 있다.

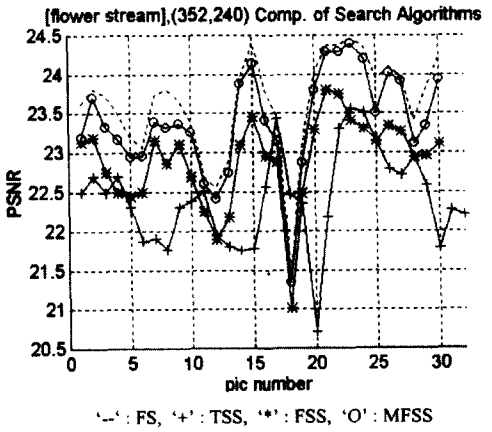


그림 2. 'Flower'영상에 대한 성능 비교
Fig. 2. Performance comparisons for the 'Flower' sequence.

표 2. 다른 종류의 영상에 대한 평균 PSNR (dB)
Table 2. Average PSNR of other sequences.

Algorithm / Image	M4SS	FS	TSS	FSS
Flower	23.398	23.626	22.413	22.893
Football	22.928	23.134	22.0234	22.131
Mobile	26.243	27.426	25.384	25.513
Bicycle	24.104	24.323	23.281	23.378

Ⅲ. 움직임 추정기의 구조

그림 3에는 MFSS 알고리즘을 이용한 움직임 추정부의 전체적인 구조를 나타내었다. 전체 구조는 탐색 영역과 기준 블럭의 데이터를 저장하기 위한 SW와 DB메모리가 있고, 이에 대한 데이터 입출력을 제어하는 SW/DB 메모리 제어부, 9개의 탐색점으로 구성되는 기본 모듈에 대한 SAD(Sum of Absolute Difference) 값을 계산하기 위해 9개의 PE(Process Element)로 구성되는 Processor Array, 또 Processor Array에서 구한 SAD값 중 최소값의 위치를 구하는 PMVG, 마지막으로 움직임 추정부의 전체적인 동작 제어와 움직임 벡터를 결정하는 MEP Controller로 구성된다. 또 이들 전체 구조에 대한 입출력으로는 탐색 영역 데이터 입력인 SW, 기준 블럭 데이터 입력인 DB, 움직임 추정의 시작 신호인 START, SW와DB 데이터의 입력 동기신호인 NEXT, 그리고 탐색 영역이 화면 가장자리에 걸쳐 있음을 나타내는 BND_SIG로 구성된다.

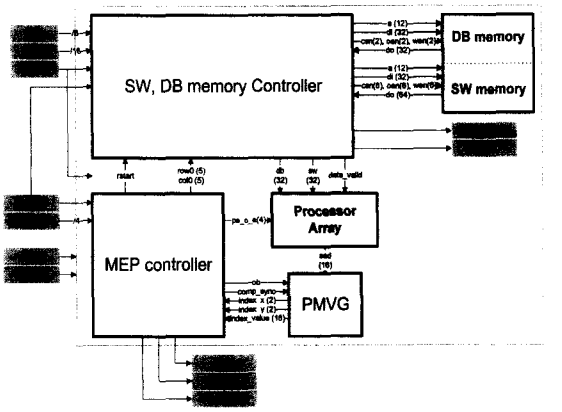


그림 3. 움직임 추정부의 전체 구조
Fig. 3. Overall architecture for the motion estimation.

1. SW 와 DB 메모리 제어부

SW와 DB 데이터는 NEXT 신호에 따라 각각의 메모리에 저장 되는데, 이때 DB와 SW의 메모리 모듈은 그림 4와 같다. 이때 외부에서 8 bit와 16 bit 단위로 입력되는 DB와 SW 데이터는 32 bit 단위로 변환되어 각 메모리에 저장되고 32 비트 단위로 읽게 된다. MEP Controller로부터 탐색 시작점 (r_0,c_0)와 rstart 신호를 받아 메모리에서 읽어내기 위한 어드레스를 생성하는데, 어드레스를 만드는 방법은 32x32 크기의 SW 데이터에서 그 탐색점의 시작점인 r0, c0를 기준으로 기본 모듈에 대한 탐색영역인 18x18의 데이터를 수평 스캔으로 순차적으로 읽도록 생성한다. SW 메모리는 32x16 크기의 메모리 3 개로 구성된다. 메모리 동작은 2 개의 메모리가 움직임 추정을 위한 탐색 영역으로 사용되는 동안 나머지 1개 메모리는 다음 블럭의 탐색영역을 저장한다. 이러한 메모리의 구성은 일반적인 이중 버퍼 구조에 비해 32x16 크기의 메모리 절약효과를 갖는다.

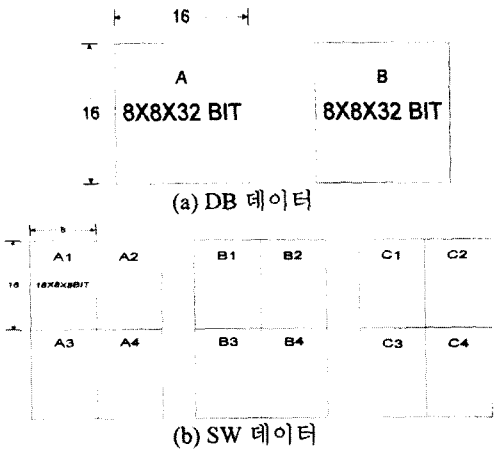


그림 4. DB/SW 메모리 모듈의 구성
Fig. 4. Architecture of DB/SW memory modules.

2. Processor Array

9개의 PE로 구성되는 Processor Array의 구조를 그림 5에 나타내었다. 움직임 추정을 위한 연산이 32 비트, 즉 4화소를 동시에 연산하기 때문에 각 PE에서의 처리를 위한 각 화소의 지연형태도 다르다. 이러한 지연형태를 포함하여 SAD(Sum of Absolute Difference) 연산을 위한 PE의 구조는 그림 6과 같고, 4 화소에 대해 병렬로 된 절대차값을 구하는 부분과 더하기, 누적기로 구성된다.

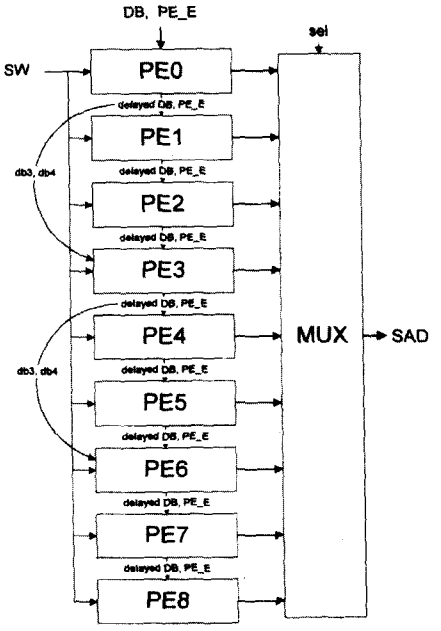


그림 5. Processor Array의 구조
Fig. 5. Processor Array architecture.

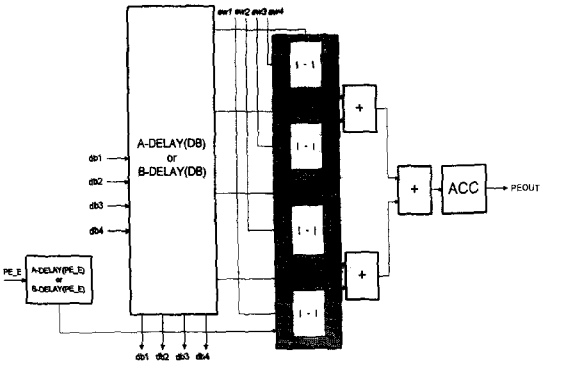


그림 6. Process Element의 구조
Fig. 6. Process Element architecture.

Processor Array를 구성하는 PE1, PE2, PE4, PE5, PE7, PE8은 같은 지연형태를 갖고, 또한 PE3와 PE6도 같은 지연형태를 포함한다. 단 PE0에는 지연 구조가 포함되지 않는다. 이러한 지연구조를 고려하여 각 PE로 입력되는 DB와 SW 데이터의 흐름도는 <표 3>과 <표 4>에 나타내었다. 이렇게 9개의 PE에서 구해진 SAD값은 비교되는 순서에 맞게 SAD값을 출력하기 위해 multiplexer가 각 PE의 출력 단에 연결된다. 이러한 Processor Array구조에서 기본 모듈의 9개 탐색점에 해당하는 SAD값을 출력하기 위해 소요되는 시간은, 4 화소를 동시에 연산하기 때문에, 기본적으로 블럭당 (16x16)/4 사이클이 필요하고, 마지막 PE인PE8까지의 DB 데이터의 지연을 합하면 총 81 사이클이 소요 된다.

표 3. SW, DB 데이터 FLOW1
Table 3. Operational flow of SW and DB data.

CLK	SW DATA				PE0				PE1				PE2				PE3				PE4			
	sw0	sw1	sw2	sw3	db0	db1	db2	db3	db0	db1	db2	db3	db0	db1	db2	db3	db0	db1	db2	db3	db0	db1	db2	db3
0	-1,-1	-1,0	0,-1	0,0	0,0	0,1	1,0	1,1		0,0		1,0							0,0	0,1				0,0
1	-1,1	-1,2	0,1	0,2	0,2	0,3	1,2	1,3	0,1	0,2	1,1	1,2	0,0	0,1	1,0	1,1			0,2	0,3		0,1	0,2	
2	-1,3	-1,4	0,3	0,4	0,4	0,5	1,4	1,5	0,3	0,4	1,3	1,4	0,2	0,3	1,2	1,3			0,4	0,5		0,3	0,4	
3	-1,5	-1,6	0,5	0,6	0,6	0,7	1,6	1,7	0,5	0,6	1,5	1,6	0,4	0,5	1,4	1,5			0,6	0,7		0,5	0,6	
4	-1,7	-1,8	0,7	0,8	0,8	0,9	1,8	1,9	0,7	0,8	1,7	1,8	0,6	0,7	1,6	1,7			0,8	0,9		0,7	0,8	
5	-1,9	-1,10	0,9	0,10	0,10	0,11	1,10	1,11	0,9	0,10	1,9	1,10	0,8	0,9	1,8	1,9			0,10	0,11		0,9	0,10	
6	-1,11	-1,12	0,11	0,12	0,12	0,13	1,12	1,13	0,11	0,12	1,11	1,12	0,10	0,11	1,10	1,11			0,12	0,13		0,11	0,12	
7	-1,13	-1,14	0,13	0,14	0,14	0,15	1,14	1,15	0,13	0,14	1,13	1,14	0,12	0,13	1,12	1,13			0,14	0,15		0,13	0,14	
8	-1,15	1,16	0,15	0,16					0,15		1,15		0,14	0,15	1,14	1,15						0,15		
9	1,-1	1,0	2,-1	2,0	2,0	2,1	3,0	3,1		2,0		3,0					1,0	1,1	2,0	2,1		1,0		2,0
10	1,1	1,2	2,1	2,2	2,2	2,3	3,2	3,3	2,1	2,2	3,1	3,2	2,0	2,1	3,0	3,1	1,2	1,3	2,2	2,3	1,1	1,2	2,1	2,2
11	1,3	1,4	2,3	2,4	2,4	2,5	3,4	3,5	2,3	2,4	3,3	3,4	2,2	2,3	3,2	3,3	1,4	1,5	2,4	2,5	1,3	1,4	2,3	2,4
12	1,5	1,6	2,5	2,6	2,6	2,7	3,6	3,7	2,5	2,6	3,5	3,6	2,4	2,5	3,4	3,5	1,6	1,7	2,6	2,7	1,5	1,6	2,5	2,6
13	1,7	1,8	2,7	2,8	2,8	2,9	3,8	3,9	2,7	2,8	3,7	3,8	2,6	2,7	3,6	3,7	1,8	1,9	2,8	2,9	1,7	1,8	2,7	2,8
14	1,9	1,10	2,9	2,10	2,10	2,11	3,10	3,11	2,9	2,10	3,9	3,10	2,8	2,9	3,8	3,9	1,10	1,11	2,10	2,11	1,9	1,10	2,9	2,10
15	1,11	1,12	2,11	2,12	2,12	2,13	3,12	3,13	2,11	2,12	3,11	3,12	2,10	2,11	3,10	3,11	1,12	1,13	2,12	2,13	1,11	1,12	2,11	2,12
16	1,13	1,14	2,13	2,14	2,14	2,15	3,14	3,15	2,13	2,14	3,13	3,14	2,12	2,13	3,12	3,13	1,14	1,15	2,14	2,15	1,13	1,14	2,13	2,14
17	1,15	1,16	2,15	2,16					2,15		3,15		2,14	2,15	3,14	3,15					1,15		2,15	

표 4. SW, DB 데이터 FLOW2
Table 4. Operational flow of SW and DB data.

CLK	SW DATA				PE4				PE5				PE6				PE7				PE8			
	sw0	sw1	sw2	sw3	db0	db1	db2	db3	db0	db1	db2	db3	db0	db1	db2	db3	db0	db1	db2	db3	db0	db1	db2	db3
0	-1,-1	-1, 0	0,-1	0, 0				0, 0																
1	-1, 1	-1, 2	0, 1	0, 2			0, 1	0, 2			0, 0	0, 1												
2	-1, 3	-1, 4	0, 3	0, 4			0, 3	0, 4			0, 2	0, 3												
3	-1, 5	-1, 6	0, 5	0, 6			0, 5	0, 6			0, 4	0, 5												
4	-1, 7	-1, 8	0, 7	0, 8			0, 7	0, 8			0, 6	0, 7												
5	-1, 9	-1,10	0, 9	0,10			0, 9	0,10			0, 8	0, 9												
6	-1,11	-1,12	0,11	0,12			0,11	0,12			0,10	0,11												
7	-1,13	-1,14	0,13	0,14			0,13	0,14			0,12	0,13												
8	-1,15	1,16	0,15	0,16			0,15				0,14	0,15												
9	1,-1	1, 0	2,-1	2, 0	1, 0			2, 0																
10	1, 1	1, 2	2, 1	2, 2	1, 1	1, 2	2, 1	2, 2	1, 0	1, 1	2, 0	2, 1	0, 0	0, 1	1, 0	1, 1		0, 0		1, 0				
11	1, 3	1, 4	2, 3	2, 4	1, 3	1, 4	2, 3	2, 4	1, 2	1, 3	2, 2	2, 3	0, 2	0, 3	1, 2	1, 3	0, 1	0, 2	1, 1	1, 2	0, 0	0, 1	1, 0	1, 1
12	1, 5	1, 6	2, 5	2, 6	1, 5	1, 6	2, 5	2, 6	1, 4	1, 5	2, 4	2, 5	0, 4	0, 5	1, 4	1, 5	0, 3	0, 4	1, 3	1, 4	0, 2	0, 3	1, 2	1, 3
13	1, 7	1, 8	2, 7	2, 8	1, 7	1, 8	2, 7	2, 8	1, 6	1, 7	2, 6	2, 7	0, 6	0, 7	1, 6	1, 7	0, 5	0, 6	1, 5	1, 6	0, 4	0, 5	1, 4	1, 5
14	1, 9	1,10	2, 9	2,10	1, 9	1,10	2, 9	2,10	1, 8	1, 9	2, 8	2, 9	0, 8	0, 9	1, 8	1, 9	0, 7	0, 8	1, 7	1, 8	0, 6	0, 7	1, 6	1, 7
15	1,11	1,12	2,11	2,12	1,11	1,12	2,11	2,12	1,10	1,11	2,10	2,11	0,10	0,11	1,10	1,11	0, 9	0,10	1, 9	1,10	0, 8	0, 9	1, 8	1, 9
16	1,13	1,14	2,13	2,14	1,13	1,14	2,13	2,14	1,12	1,13	2,12	2,13	0,12	0,13	1,12	1,13	0,11	0,12	1,11	1,12	0,10	0,11	1,10	1,11
17	1,15	1,16	2,15	2,16	1,15		2,15		1,14	1,15	2,14	2,15	0,14	0,15	1,14	1,15	0,13	0,14	1,13	1,14	0,12	0,13	1,12	1,13

3. PMVG

PMVG에서는MFSS 알고리즘의기본 모듈에 해당하는 9개 탐색점에 대한 SAD 출력을 MEP Controller로부터 동기 신호를 받아 순서대로 비교한다. 따라서기본 모듈의 9개 탐색점 중에 최소 에러점을 구하기 위해서 동기 신호가 9번 enable 된다. 정해진 순서에 따라 비교되고 이 중에서 최소 에러점에 대한 위치와 값을 MEP_Controller로 출력한다.

4. MEP Controller

MEP Controller는 움직임 추정기의 전체적인 동작을 제어하는데 하는 일을 몇 가지로 요약하면 다음과 같다. 첫째, SW와 DB 데이터를 읽는 시작신호와 SW의 탐색 시작점 어드레스를 생성하는 일이다. 둘째, 9개 PE의 결과에 대해 최종 출력할 시간을 알려주는 신호를 생성한다. 셋째, PMVG가 9개의 SAD 출력 값에서 최소값을 구하는 비교를 하기 위한 동기 신호를 생성한다. 이렇게 하여 움직임 추정의 최종단계를 거쳐 움직임 벡터가 계산되면 수평과 수직 움직임 벡터와 동기신호를 출력한다. MEP_Controller의 동작 제어는 내부 카운터로 이루어지는데 추정 시작 신호에 동기를 맞춰 시작되고 최종 움직임 벡터가 출력되면 초기화 된다. 전체적인 동작 순서는 MFSS 알고리즘의 스텝을 세는 스텝카운터가 0부터 3까지 증가하고 각 스텝카운터 값에 대해 다시 기본 모듈을 세는 모듈카운터가 0부터 3까지 증가한다. 모듈카운터가 변할 때마다 기본 모듈에서의 최소 에러값들을 비교하고 SW와 DB 메모리 제어부에 읽어낼 데이터의 시작 어드레스를 출력한다.

Ⅳ. 움직임 보상기의 구조

그림 7은 움직임 보상기의 전체 구조에 관한 블럭도이다. 각 블럭을 살펴보면 Y 영상에 대해 움직임 보상 하는 부분인 MC_Y, U와 V에 대해 움직임 보상하는 부분인 MC_UV, 그리고 이들 보상된 데이터의 출력을 선택하는 MC_MUX가 있다. 입출력은 움직임 추정기로부터 출력하는 Y에 대한 보상 데이터 입력으로 SWY0, SWY1, SWY2, SWY3이 있고, U와V 영상에 대한 SW 데이터 입력으로 SW, 움직임 벡터 출력의 동기 신호인 MV_SYNC, 움직임 벡터인 MVX와 MVY, 움직임 보상 시작을 알리는 START, 마지막으로 움직임 보상되어 출력되는 Mced 신호로 구성된다. 움직임 추정은 Y성분에 대해서만 적용한 반면, 움직임 보상에서는 Y, U, V 영상에 대해 각기 적용된다. Y,U,V가 4:2:0 포맷으로 샘플링 되었을 경우 U와 V 의 움직임 보상에서는 Y 움직임 벡터를 2로 나누어 사용한다.

1. 움직임 추정기와 보상기의 결합

제안한 움직임 추정 및 보상기는 기존에 움직임 추정기와 보상기를 분리하였던 것을, 탐색 영역을 공유할수 있도록 움직임 추정기와 보상기가 결합된 구조를 갖는다. 기존과 같이 분리되어 있을 때에는 같은 SW 데이터를 2번 사용해야 하기 때문에 메모리의 낭비 요인이 된다. 그러나 결합되면 움직임 추정을 위한 SW 데이터와 움직임 보상된 데이터인 16x16의 데이터만 사용되어 16x16x3의 크기만큼의 메모리가 절감되고 출력이 나오기까지 전체적인 지연시간도 줄일수 있다.

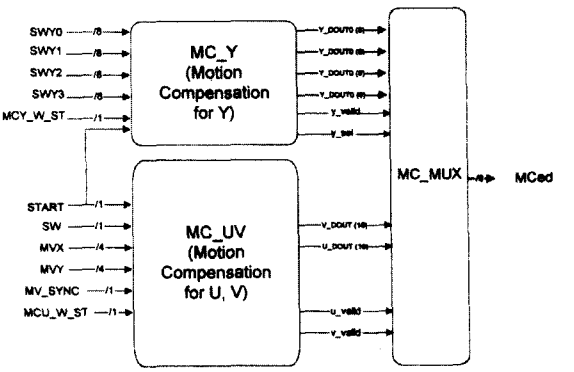


그림 7. 움직임 보상부의 전체 구조
Fig. 7. Overall architecture for the motion compensation.

2. Y에 대한 움직임 보상기

Y 에 대한 움직임 보상기는 그림 8과 같이 8x8의 메모리 모듈 4개와 메모리 모듈의 입출력을 제어하는 RW_CNTL로 구성된다. Y에 대한 움직임 보상은 움직임 추정기에서 움직임 벡터를 구하는 과정의 마지막 스텝에서 움직임 벡터가 가리키는 곳의 16x16 데이터를 미리 읽어 오기 때문에 Y에 대한 움직임 보상기에서는 Y에 대한 SW 데이터를 저장할 필요가 없다. 따라서 MC_Y가 하는 일은 움직임 추정부에서 출력하는 움직임 보상된 데이터를 저장하고 있다가 움직임 보상 시작 신호인 start 신호에 맞춰 움직임 보상된 Y 데이터를 해당하는 속도로 출력한다. MC_Y의 메모리 구성은 움직임 추정기에서 움직임 보상된 데이터가 32비트 단위이므로 8 비트 단위 메모리 모듈 4개를 이용하여 메모리에 읽고 쓰기를 32비트로 하고, 32비트로 출력되는 데이터에서 해당하는 8비트만 선택하도록 하였다.

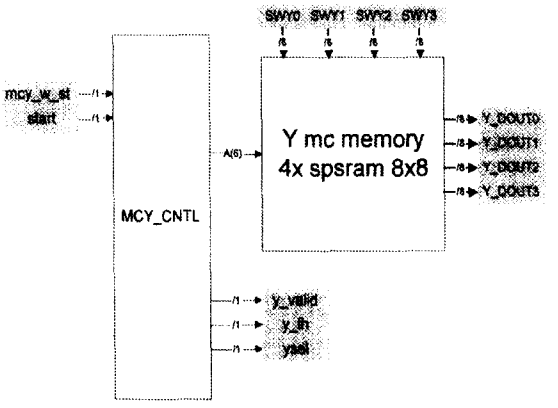


그림 8. MC_Y의 내부 블럭도
Fig. 8. Block diagram for the MC_Y.

3. U,V 신호의 움직임 보상기

U와 V에 대한 움직임 보상기의 구조는 그림 9에 나타내었고 16x8 크기의 SW 메모리 2개로 구성된다. U와 V에 대한 SW 데이터의 입력 형태는 U와 V가 각 8비트씩 16 비트 버스를 통해 동시에 입력된다. 따라서 U, V에 대한 쓰기 어드레스는 같은 어드레스를 동시에 만들고 데이터를 8비트씩 나누어 저장하게 된다. <표 5>는 U와 V에 공통적으로 적용되는 각 메모리의 입출력 상태의 흐름을 나타내었다. 본 논문에서는 한 매크로블럭 주기 동안에 읽고 쓰는 동작을 시간 분할로 나누어 처리함으로써 소요되는 메모리의 양을 반으로 줄였다. 메모리에 입력되는 탐색 영역을 저장한 후에 움직임 보상을 위한 탐색 영역이 준비되면, 이들 메모리로부터 읽어내어 움직임 보상된 데이터를 출력한다. 매크로 블럭의 주기를 8 블럭(8x8 크기) 구간인 512 사이클로 하였을경우 전체적인Y, U, V에 대한 입출력의 시간도는 그림 10과 같다.

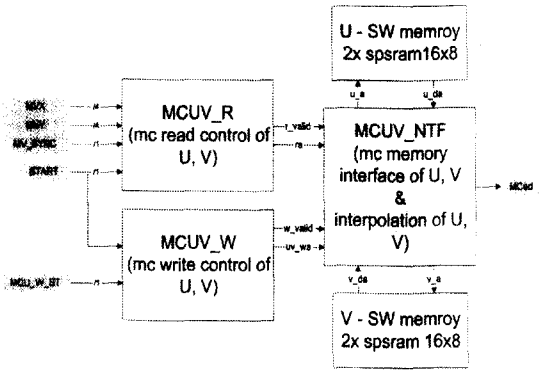


그림 9. MC_UV의 구조
Fig. 9. MC_UV architecture.

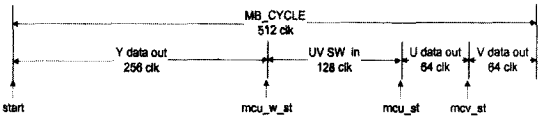


그림 10. Y, U, V에 대한 입출력 타이밍도
Fig. 10. Timing diagram of the Y,U,V I/O signal.

표 5. 색차신호 SW 메모리의 입출력 상태
Table 5. Status of chrominance SW memory.

MB cycle \ 블럭(16x8)	Left Block	Right Block
0	W R	x R
1	x R	W R
⋮	위의 형태의 반복	위의 형태의 반복

V. 설계 결과

본 설계는 탑다운 설계 방식으로 VHDL를 이용하여 설계하였다. 합성 툴은 Synopsys사의 Design Analyzer를 사용하였고 모의실험은 VSS를 사용하였다. ASIC을 위한 Target Library는 SAMSUNG의 Standard Cell인 STD70(0.6m)를 사용하였다.

<표 6>은 본 논문에서 제안한 움직임 추정기의 설계 결과를 LSI Logic사의 L64720 움직임 추정기 칩과[19] 비교한 것이다. L64720 에서 사용했던 SW

데이터의 저장방식인 이중 버퍼 구조대신 회전 구조의 메모리를 사용하여 512 byte의 메모리를 줄일 수 있다. 또 움직임 벡터를 구하는데 필요한 사이클 수도 약 3배 정도의 차이를 보인다. 때문에 약 30 Mhz의 클럭 속도에서 L64720는 352x240의 영상 크기에 대해서만 실시간으로 동작하지만 제안된 구조에서는 720x480크기의 영상에 대해서도 실시간으로 움직임 추정하는 성능을 갖는다. 또한 게이트수도 약 10,000 게이트 이상 줄어든 것을 알수 있다.

표 6. 움직임 추정기의 사양 비교
Table 6. Comparisons of motion estimators.

	L64720	Proposed	
Algorithm	FS	MFSS	
Memory	2560 byte	2048 byte	
Required cycle(Delay)	2237 cycle	882 cycle	
Clock frequency	20 Mhz	13.5Mhz	27 Mhz
Frame size for real time	352x240	352x240	720x480
Motion Vector Range	-8~7	-8~7	
No. of Gates	>30,000	<20,000	
Search Area	32x32	32x32	

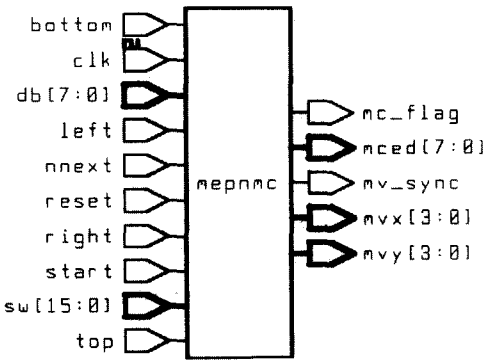


그림 11. 움직임 추정 및 보상기의 symbol
Fig. 11. Synthesized symbol for the motion estimation and compensation.

그림 11은 Synopsys의 Design Analyzer를 이용하여 합성한 결과의 상위 블럭의 입출력을 나타내었다. 입력 포트에는 clk, reset, start와 함께 SW, DB데이터, 그리고 경계영역을 나타내는 bottom, left, right, top이 있고, 출력 포트에는 움직임 벡터인 mvx와 mvy, MC, no MC를 알리는 mc_flag와 움직임 보상된 데이터 mced가 있다. 요구되는 전체 메모리 양은 2816 btye로 움직임 추정기에서 2048byte가 필요하고 움직임 보상기에서 768byte가 사용된다. 또 전체 게이트수는 메모리를 제외한 로직이 약 28,000 게이트 정도이다.

VI. 결 론

본 논문에서는 기존의 고속 움직임 추정기의 단점을 보완하고 VLSI 구현에 적합한 MFSS 알고리즘을 제안하였다. 모의실험을 통해 기존의 고속 움직임 추정 알고리즘에 비해 전반적으로 우수한 성능을 보임을 알 수 있었다. MFSS 알고리즘은기존의 고속 탐색 알고리즘에 비해 일정한 규칙을 갖기 때문에 하드웨어 구현에 적합한 알고리즘이다. 또한, 본 논문에서 소개한 MFSS를 이용하여 움직임 추정기 및 보상기의 VLSI 구조를 제안하였고, VHDL을 이용하여 설계한 결과를 제시하였다. 제안한 알고리즘을 이용한 VLSI 설계는 동일한 사양의 움직임 추정기와 보상기에 비해 메모리 및 로직 크기를 어느 정도 최적화하였고, 실시간 구현에 중요한 요소인 출력이 나오기 까지의 지연시간도 많이 줄이었다.

앞으로 고해상도의 영상을 위한 추정 알고리즘 및 VLSI 구조에 대한 연구가 수행되어야 하겠다.

참 고 문 헌

[1] H. M. Musmann, P. Pirsch, and H. J. Gravoert, "Advances in picture coding," *Proc. IEEE*, vol. 73, pp. 523-548, April 1985.

[2] H. Gharavi and Mike Mills, "Blockmatching motion estimation algorithms-new results," *IEEE Trans. Circuits Syst.*, vol. 37, no. 5, pp. 649-651, May 1990.

[3] M. Bierling, "Displacement estimation by hierarchical blockmatching," *SPIE*, vol.1001, Visual Communications and Image Processing, 1988.

[4] Lai-Man Po and Wing-Chung Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 3, p. 313-317, June 1996.

[5] M. J. Chen, L. G. Chen, T. D. Chiueh, and Y. P. Lee, "A new block-matching criterion for motion estimation and its implementation," IEEE Trans. Circuits

Syst. Video Technol., vol. 5, no. 3, pp. 231-236, June 1995.

[6] M. J. Chen, L. G. Chen, and T. D. Chiueh, "One-dimensional full search motion estimation algorithm for video coding." *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, no. 5, pp. 504-509, Oct. 1994.

[7] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, no. 4, pp. 438-442, Aug. 1994.

[8] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, no. 5, pp. 438-442, April 1993.

[9] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Trans. Commun.*, vol. 38, no. 7, pp. 950-953, July 1990.

[10] L. W. Lee, J. F. Wang, J. Y. Lee, and J. D. Shie, "Dynamic search window adjustment and interlaced search for block-matching algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, no. 4, pp. 85-87, Feb. 1993.

[11] T. Komarek and P. Pirsch, "Array architectures for block-matching algorithms," *IEEE Trans. Circuits Syst.*, vol. 36, no. 10, pp. 1301-1308, Oct. 1989.

[12] S. B. Pan, S. S. Chae, and R. H. Park, "VLSI architectures for block matching algorithms using systolic arrays," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 1, pp. 67-73, Feb. 1996.

[13] L. D. Vos and M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, no. 10, pp.1309-1316, Oct. 1989.

[14] C. H. Hsieh and T. P. Lin, "VLSI architecture for block-matching motion estimation algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 169-175, June 1992.

[15] S. Dutta and W. Wolf, "A flexible parallel architecture adapted to block-matching motion-estimation algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 1, pp. 74-86, Feb. 1996.

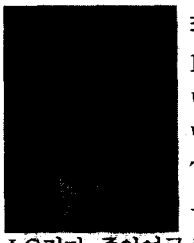
[16] H. Jong, L. Chen, and T. Chiueh, "Parallel architectures for 3-step hierarchical search block-matching algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, no. 4, pp. 407-416, Aug. 1994.

[17] Y. S. Jehng and L. G. Chen, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. Signal Process.*, vol. 41, no. 2, pp. 889-900, Feb. 1993.

[18] K. Yang, M. Sun, and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," IEEE Trans. Circuits Syst., vol. 36, no. 10, pp. 1317-1325, Oct. 1989.

[19] LSI Logic CCITT Video Compression Databook, L64720 Motion Estimation Processor, LSI Logic, 1989.

저 자 소 개



李 東 潞(正會員)

1986년 한양대학교 전자공학(공학사). 1988년 미국 The Univ. of Texas at Austin 전기 및 컴퓨터 공학과(공학석사). 1991년 미국 The Univ. of Texas at Austin 전기 및 컴퓨터 공학과(공학박사). 1991년 ~ 1994년 LG전자 중앙연구소 선임연구원. 1994년 ~ 현재 한양대학교 제어계측공학과 조교수. 주관심분야는 영상처리 및 압축, 디지털 시스템, VLSI 설계