論文95-32B-12-9

16:1 부분 표본 추출 블럭 정합 알고리즘과 이의 하드웨어 설계

(A 16:1 Subsampling Block-Matching Algorithm and Its Hardware Design)

金 良 勳 *, 林 鍾 錫 **, 閔 丙 基 ***

(Yang Hoon Kim, Chong Suck Rim, and Byoungki Min)

요 약

동작 예측을 위한 기존의 전역 검색 블럭 정합 알고리즘(Full Search Block Matching Algorithm(FSA))은 전체 계산량이 과도하기 때문에 이의 하드웨어 구현을 위한 비용이 매우 크다. 본 논문에서는 16:1 부분 표본 추출 방법을 이용한 효율적인 블럭 정합 알고리즘과 이의 하드웨어 구조를 제안한다. 이 방법은 검색 위치의 수를 줄이는 대신, 각 검색 위치에서 두 화소 차이의 평균을 계산하는데 이용되는 화소의 수를 줄이므로써 계산량을 절감하는 방법이다. 이 방법은 Liu과 Zaccarin이 제안한 4:1 부분 표본 추출에 의한 전역 검색 블럭 정합 알고리즘를 확장한 방법으로 FSA에 비해 계산량은 16분의 1로 줄지만, FSA와 거의 유사한 성능을 갖는다. 또한 이 알고리즘을 구현하기 위하여 본 논문에서 제안한 하드웨어는 MPEG-I 부호기의 실기간 동작 예측에 사용할 수 있다.

Abstract

Conventional full search block matching algorithm for motion estimation is computationally intensive and the resulting hardware cost is very high. In this paper, we present an efficient block matching algorithm using a 16:1 subsampling technique, and describe its hardware design. The algorithm reduces the number of pixels in calculating the mean absolute difference at each search location, instead of reducing the search locations.The algorithm is an extension of the block mating algorithm with 4:1 subsampling proposed by Liu and Zaccarin such that the amount of computation is reduced by a factor of 4(16 compared to the full search block matching algorithm) while producing similar performance.The algorithm can efficiently be designed into a hardware for real-time applications.

* 正會員, LG電子 情報 시스템 硏究所 PC 硏究室
(PC Lab. Information Systems R&D Center, LE Elec.)
** 正會員, 西江大學敎 電子計算學科
(Dept. of Computer Science, Sogang Univ.)
*** 正會員, 韓國電子通信 硏究所 미디어 硏究室
(Media Res. Lab., Electronics and Telecommunications Research Institute)
※ 이 연구는 1994 년도 한국전자통신연구소 연구비 지원에 의한 결과임.
接受日字:1994年11月21日, 수정완료일:1995年12月6日

I. 서 론

디지탈 텔레비젼이나 비디오폰 등 동영상 전송이 요구되는 응용에 있어서 데이타 압축은 필수적이다. 동영상을 압축하는 데는 동작 예측/보상, DCT(Discrete Cosine Transform), VLC(Variable Length Coding) 등 두 가지 이상의 압축 방법을 복합 적용한 하이브리드 부호화 방법이 가장 보편적으로 사용되며 MPEG(Moving Pictures Export Group)-I,II 등 이의 표준안이 제안되었다[3,6,9]. 하이브리드 부호화 방법에 사용되는 방법들중 동작 예측/보상 방법은 연속적인 영상들간에 존재하는 시간적 중복을 제거하여

동영상을 압축하는 방법이다.

동작 예측/보상 방법에서 블럭 정합 알고리즘을 이용한 방법은 비록 계산량은 과다하지만 계산 방법이 단순하며 일정하여 많이 사용되고 있다[10]. 블럭 정합 알고리즘은 현재 영상을 일정한 크기의 블럭들로 분할하여, 참조 영상에서 각 현재 영상 블럭에 가장 잘 정합되는 블럭을 찾는 방법이다. 블럭 정합 알고리즘을 사용하여 참조 영상에서 찾은 블럭과 이의 대응되는 현재 영상 블럭간의 위치 차를 현재 영상 블럭에 대한 움직임 벡터라 하며, 동작 예측/보상 방법은 이러한 움직임 벡터와 이에 대응되는 두 블럭간의 차이를 이용하여 동영상을 높은 비율로 압축할 수 있도록 한다.

연속적인 영상들사이에서 블럭들의 이동은 작다고 가정할 수 있으므로, 계산량을 줄이기 위해 움직임 벡터의 각 성분 x,y축에 대한 최대 크기를 제한한다. 이러한 제한에 따라 그림 1에 보인 바와 같이, 각 현재 영상 블럭의 움직임 벡터 계산에 사용되는 참조 영상의 화소로 구성된 영역을 현 영상 블럭에 대한 검색 영역이라고 한다. 또한 블럭의 위치는 블럭이 존재하는 영상에서 블럭의 가장 왼쪽 상단 화소 위치로 정의하며, 검색 영역에서 정합 계산에 사용되는 블럭의 위치를 검색 위치라고 한다. 그리고 현재 영상 블럭과 검색 위치간에 위치 값들의 차이를 편의상 DCS(*Displacement between the two locations of the Current block and the Search location*)이라고 한다.

두 블럭 간의 정합 계산을 위하여 여러 방법이 존재한다[10]. 이중 두 화소 차이의 평균(*Mean Absolute Difference*(MAD)) 정합 계산 방법은 두 블럭을 구성하는 화소의 차이의 합으로써 단순한 연산자로 정의되어 쉽게 하드웨어로 구현할 수 있으며 이의 성능도 타당하기 때문에, 정합 계산 방법으로 가장 널리 사용되고 있으며[7] 본 논문에서 제안하는 방법도 MAD 정합 계산 방법을 사용한다.

블럭과 검색 영역의 크기를 각각 $M\times N$과 $m\times n$이라고 정의하면 현재 영상의 한 블럭에 대해 $(m-M+1)(n-N+1)$개의 검색 위치가 존재한다. 전역 검색 블럭 정합 알고리즘(*Full Search Block Matching Algorithm*(FSA))은 블럭 정합 알고리즘 중 가장 최적의 움직임 벡터를 계산하는 방법으로 하나의 현재 영상 블럭에 대해 검색 영역내에 존재하는 모든 검색 위치에서MAD를 계산하여 이중 가장 작은 MAD를 갖는 검색 위치를 선택하여 이의 DCS를 움직임 벡터로 한다. 그러나 FSA는 그 계산량이 과도해서 실시간 응용을 위한 하드웨어 비용이 매우 크기때문에[7,12,13] 이를 줄이기 위해 *Three Step Search*(TSS)[2], 2D logarithmic search[5], Conjugate search[11], *Hierarchical Block Matching Algorithm*(HBMA)[1]등 많은 블럭 정합 알고리즘이 제안되었다. 제안된 대부분의 방법들은 가장 최적인 검색 위치와 거리의 차가 작은 검색 위치일수록 정합의 오차가 작다는 가정하에 검색 위치의 수를 제한한다. 따라서 지역적 최적값에 빠지기 쉬운 특성을 가지므로, FSA에 비해 성능이 상당히 떨어지는 단점을 갖는다. 또한 이들 방법은 비교적 많은 단계를 통해 움직임 벡터를 계산하는데, 각 단계의 계산 결과에 따라 다음 단계에서 MAD를 계산할 검색 위치가 결정되는 제어상의 불규칙성때문에 병렬성이 떨어지며, 따라서 하드웨어 설계에 많은 어려움이 따른다.

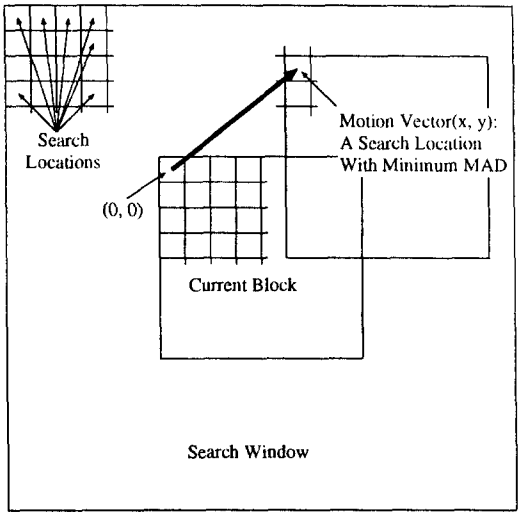

그림 1. 블럭 정합 알고리즘에서의 기본 정의.
Fig. 1. Basic definition in block matching algorithm.

최근, Liu과 Zaccarin은 교대적인 4:1 부분 표본 추출 방법을 이용한 전역 검색 블럭 정합 알고리즘(*Full Search Block Matching Algorithm With Alternating 4:1 Subsampling*(편이상 이를 4:1 FSA라 언급))을 제안하였다[8]. 이 방법은 검색 위치의 수를 줄이는 대신, 각 검색 위치에서 MAD 계산에 사용되는 화소의 수를 제한함으로써 계산량을 감소하는 방법이다. 4:1 FSA의 성능은 FSA와 유사하면서 전체 계산량을 약 4분의 1로 감소시킨다. 하지만, 4:1

FSA를 실시간 응용에 적용하기에는 여전히 많은 계산량이 필요하며, 또한 이의 하드웨어 구현도 쉽지 않다.

본 논문에서는 4:1 FSA를 확장하여 16:1 부분 표본 추출에 의한 새로운 블럭 정합 알고리즘과 이의 하드웨어 구조를 보인다. 제안한 알고리즘의 계산량은 FSA에 비해 약 16분의 1에 불과하며 이의 알고리즘은 4:1 FSA뿐만 아니라 FSA와 유사한 성능을 가진다. 또한 TSS와는 달리 제어상의 불규칙성이 적기 때문에 실시간 처리를 위한 하드웨어 구현이 용이하며, 계산량을 많이 줄였기 때문에 이의 하드웨어는 단일 칩으로 MPEG-I 부호기 시스템의 동작 예측기로 사용될 수 있다. 본 서론에 이어 다음 II장과 III장에서는 우리의 블럭 정합 알고리즘과 이의 하드웨어 구조를 각각 설명하고 마지막 IV 장에서 결론을 내린다.

II. 16:1 부분 표본 추출 블럭 정합 알고리즘

1. 알고리즘

p번째 영상에서 좌표 (x, y)의 화소값을 $F_p(x, y)$로 표시하고 $n, n-1$번째 영상을 각각 현재 영상, 참조 영상으로 했을때 현재 영상 블럭 (k, l)의 DCS (x, y)에 해당되는 검색 위치에서 MAD를 다음의 식으로 계산한다.

$$MAD_{(k,l)}(x,y) = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}|F_n(k+i, l+j) - F_{n-1}(k+x+i, l+y+j)|$$

대부분의 블럭 정합 알고리즘은 위의 식을 사용해서 검색 위치에서 현재 영상 블럭과의 정합을 계산하므로, 현재 영상 블럭을 구성하는 화소와 검색 영역에서 이에 대응되는 모든 화소를 정합 계산에 사용하였다. 그러나 블럭 정합 알고리즘은 블럭의 모든 화소가 같은 거리만큼 이동한다는 가정에 따라 제안되었고 영상 특성상 각 화소는 인접한 화소와 서로 유사한 값을 가지므로, 블럭의 일부 화소만 사용하여 정합 계산을 하더라도, 비교적 정확한 움직임 벡터를 구할 수 있다. 하지만 너무 적은 수의 화소 또는 블럭내에서 적절히 분산되지 않은 화소를 사용한다면 블럭의 전체적인 특징을 반영하지 못하므로 이의 결과는 만족스럽지 못할 것이다. 이러한 사실에 기반을 두어, 이 장에서는 교대적인 16:1 부분 표본 추출 방법을 이용한 전역 검색 블럭 정합 알고리즘(*Full Search Block Matching Alogorithm with Alternating 16:1 Subsampling*(편의상 이를 16:1 FSA로 언급))을 제시한다.

0	4	8	12	0	4	8	12
5	1	13	9	5	1	13	9
10	14	2	6	10	14	2	6
15	11	7	3	15	11	7	3
0	4	8	12	0	4	8	12
5	1	13	9	5	1	13	9
10	14	2	6	10	14	2	6
15	11	7	3	15	11	7	3

그림 2. 현재 영상 블럭의 화소 레이블링.
Fig. 2. Pixel labeling in a current frame block.

그림 2에 보인 것처럼 현재 영상 블럭의 화소에 0부터 15까지 규칙적으로 레이블을 부여한다. 레이블 i가 부여된 화소의 집합을 P_i라 표시하며 이를 화소 그룹이라 부른다. 따라서 총 16 개의 화소 그룹이 정의된다. 검색 위치에서 임의의 한 화소 그룹과 검색 영역에서 이에 대응되는 화소를 사용하여 계산한 정합 계산값을 부분 MAD(*Partial MAD*(PMAD))라 언급하며, 화소 그룹 P_i를 사용해 계산한 PMAD를 $PMAD_i$로 표시한다. 만일 모든 검색 위치에서 단 하나의 화소 그룹만을 사용하여 PMAD를 계산하면, 즉 화소 그룹 P_0만을 사용하면 현재 영상 블럭 (k, l)의 DCS (x, y)에서 $PMAD_0$를 다음의 식으로 계산한다.

$$PMAD_{0(k,l)}(x,y) = \frac{16}{MN}\sum_{i=0}^{\frac{M}{4}-1}\sum_{j=0}^{\frac{N}{4}-1}|F_n(k+4i, l+4j) - F_{n-1}(k+x+4i, l+y+4j)|$$

만일 모든 검색 위치에서 PMAD를 계산한 후, PMAD를 최소로 하는 검색 위치의 DCS를 움직임 벡터로 선택하는 방법은 위의 식에 보인바와 같이 각 검색 위치에서 정합 계산량을 16분의 1로 줄이기 때문에 전체 계산량은 FSA보다 16분의 1로 감소한다. 하지만 나머지 다른 화소 그룹들은 움직임 벡터를 찾는데 전혀 사용되지 않았기 때문에, 이의 움직임 벡터는 만족스럽지 않다. 이러한 결점을 피하기 위해, 검색 위치에서 사용하는 화소 그룹을 골고루 사용하여 PMAD를 계산하는데 이는 다음과 같이 정의된다.

검색 위치에 그림 3에 보인 바와 같이 0부터 15까지 레이블을 규칙적으로 부여한다. 검색 위치에 부여된 레이블은 그 검색 위치에서 PMAD 계산에 사용하는 화소 그룹을 지정한다. 따라서 레이블 $i(=0, 1, 2, \cdots, 15)$가 부여된 검색 위치에서는 화소 그룹 P_i를 사용하여 그 검색 위치에서 $PMAD_i$를 계산한다. 만일 화소 그룹

P_0의 화소가 DCS (x, y)에서 사용되었다고 가정하자. 그러면 나머지 DCS의 PMAD 계산에 사용하는 화소 그룹은 표 1에 보인 바와 같다. 이 표에서 i, j 는 정수값이며 위의 표에 나타낸 DCS가 검색 영역내에 존재하도록 하는 범위를 가진다. 또한 DCS가 (x, y)인 검색 위치에서 현재 영상 블럭 (k, l)에 대해 화소 그룹 사용에 따른 PMAD는 표 2에 보인다.

0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15
0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15

그림 3. 검색 영역의 화소 레이블링.
Fig. 3. Labeling of the search locations.

표 1. 각 검색 위치에서 화소 그룹의 사용.
Table 1. Pixel group used at each search location.

DCS	Pixel Group
$(x+4i,\ y+4j)$	P_0
$(x+4i+1,\ y+4j)$	P_1
$(x+4i+2,\ y+4j)$	P_2
$(x+4i+3,\ y+4j)$	P_3
$(x+4i,\ y+4j+1)$	P_4
$(x+4i+1,\ y+4j+1)$	P_5
$(x+4i+2,\ y+4j+1)$	P_6
$(x+4i+3,\ y+4j+1)$	P_7
$(x+4i,\ y+4j+2)$	P_8
$(x+4i+1,\ y+4j+2)$	P_9
$(x+4i+2,\ y+4j+2)$	P_{10}
$(x+4i+3,\ y+4j+2)$	P_{11}
$(x+4i,\ y+4j+3)$	P_{12}
$(x+4i+1,\ y+4j+3)$	P_{13}
$(x+4i+2,\ y+4j+3)$	P_{14}
$(x+4i+3,\ y+4j+3)$	P_{15}

모든 검색 위치에서 PMAD 계산을 완결한 후, 각 화소 그룹 $i(=0,1,\cdots,15)$에 대해 PMAD_i를 최소로 하는 검색 위치(이러한 DCS를 후보 움직임 벡터라 언급)를 선택한다(따라서 총 16 개의 후보 움직임 벡터가 선택됨). 그리고 나서 각 후보 움직임 벡터가 가리키는 검색 위치에서 현재 영상 블럭의 전체 화소를 사용하여 MAD를 계산하고, 이중 MAD를 최소로 하는 검색 위치를 가리키는 후보 움직임 벡터를 움직임 벡터로 선택한다. 그러나 화소 그룹을 구성하는 화소의 수가 블럭을 구성하는 화소의 수보다는 상대적으로 너무 적기 때문에, 이 방법은 정확하지 않은 움직임 벡터를 계산해 낼 수 있다. 이는 같은 화소 그룹을 사용하여 PMAD를 계산한 두 검색 위치 L_0, L_1에 대해서, L_0에서 PMAD가 L_1에서 PMAD보다 작지만 L_1에서의 MAD가 L_0보다 작은 경우가 존재하기 때문이다. 이러한 단점을 보완하기 위해, 각 화소 그룹에 대해 PMAD를 최소로 하는 2 개의 후보 움직임 벡터를 선택(따라서 총 32 개의 후보 움직임 벡터가 선택됨)하여, 위에 설명한 방법에 의해 이중에서 MAD를 최소로 하는 검색 위치를 가리키는 후보 움직임 벡터를 움직임 벡터로 선택한다.

표 2. 화소 그룹 사용에 따른 PMAD.
Table 2. The PMAD according to using pixel group.

Pixel grooup	PMAD
P_0	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i,l+4j)-F_{n-1}(k+x+4i,l+y+4j)\rvert$
P_1	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+1,l+4j+1)-F_{n-1}(k+x+4i+1,l+y+4j+1)\rvert$
P_2	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+2,l+4j+2)-F_{n-1}(k+x+4i+2,l+y+4j+2)\rvert$
P_3	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+3,l+4j+3)-F_{n-1}(k+x+4i+3,l+y+4j+3)\rvert$
P_4	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+1,l+4j)-F_{n-1}(k+x+4i+1,l+y+4j)\rvert$
P_5	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i,l+4j+1)-F_{n-1}(k+x+4i,l+y+4j+1)\rvert$
P_6	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+3,l+4j+2)-F_{n-1}(k+x+4i+3,l+y+4j+2)\rvert$
P_7	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+2,l+4j+3)-F_{n-1}(k+x+4i+2,l+y+4j+3)\rvert$
P_8	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+2,l+4j)-F_{n-1}(k+x+4i+2,l+y+4j)\rvert$
P_9	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+3,l+4j+1)-F_{n-1}(k+x+4i+3,l+y+4j+1)\rvert$
P_{10}	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i,l+4j+2)-F_{n-1}(k+x+4i,l+y+4j+2)\rvert$
P_{11}	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+1,l+4j+3)-F_{n-1}(k+x+4i+1,l+y+4j+3)\rvert$
P_{12}	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+3,l+4j)-F_{n-1}(k+x+4i+3,l+y+4j)\rvert$
P_{13}	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+2,l+4j+1)-F_{n-1}(k+x+4i+2,l+y+4j+1)\rvert$
P_{14}	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i+1,l+4j+2)-F_{n-1}(k+x+4i+1,l+y+4j+2)\rvert$
P_{15}	$\frac{16}{MN}\sum_{i=0}^{M/4-1}\sum_{j=0}^{N/4-1}\lvert F_n(k+4i,l+4j+3)-F_{n-1}(k+x+4i,l+y+4j+3)\rvert$

표 1에 보인 화소 그룹의 선택 방법은 현재 영상 블럭과 검색 영역의 화소를 모두 균등하게 사용함을 보장한다. 따라서 블럭의 전체적 특징을 잘 반영하므로 보다 정확한 움직임 벡터를 얻을 수 있다.

하나의 현재 영상 블럭의 움직임 벡터를 계산하기 위한 16:1 FSA의 계산량은 다음과 같이 산출된다. K

를 두 화소의 차를 계산하고 이를 누적하는데 필요한 계산수라고 하자. FSA는 각 검색 위치에서 KMN의 계산이 필요하므로, 한 블럭의 움직임 벡터를 찾기 위해 총 $K(m-M+1)(n-N+1)MN$ 개의 계산이 필요하다. 16:1 FSA는 각 검색 위치에서 $KMN/16$ 개의 계산이 필요하므로, 32 개 후보 움직임 벡터를 찾기 위해 총 $K(m-M+1)(n-N+1)MN/16$ 개의 계산이 필요하다. 추가로 32 개 후보 움직임 벡터중 하나를 선택하기 위해 32 개 후보 움직임 벡터가 가리키는 검색 위치에서 현재 영상 블럭의 전체 화소를 이용해서 MAD를 계산해야 하므로 이에 $32\frac{15}{16}KMN$ 개의 계산이 필요하다. 여기서 $30KMN \ll K(m-M+1)(n-N+1)MN/16$일 경우, 16:1 FSA는 FSA에 비해 계산량을 약 16분의 1로 줄인다.

2. 실험 결과

CIF 규격(352×240 pixels/frame, 30 frames/sec)의 영상에 대해서 검색 영역과 블럭의 크기를 각각 47×47과 16×16으로하고 본 논문에서 제안한 알고리즘을 평가하였다. 실험에 사용한 영상은 'Football'의 59 장 영상, 'Table tennis', 'Flower Garden'의 101 장 영상등이다.'Flower Garden'은 물체가 고정된 채 카메라가 한 방향으로 움직이면서 찍은 영상이고, 'Table tennis'는 점차 물체가 축소되며 갑자기 장면 변환이 있는 영상이며, 'Football'은 물체들이 독립적으로 매우 큰 움직임을 가진 영상이다. 이러한 영상을 대상으로 먼저 연속적인 영상사이에서 각 블럭 정합 알고리즘의 검색 능력을 비교하였다. 즉 n 번째 영상을 참조 영상, $n+1$ 번째 영상을 현재 영상으로 하여 각 블럭 정합 알고리즘에 대해 움직임 벡터를 찾은 후, 원래 영상과 오차를 PSNR(*Peak Signal to Noise Ratio*)로 구하여 비교하였다.

첫 실험은 16:1 FSA에서 각 화소 그룹에 대해 후보 움직임 벡터를 각각 1, 2, 4 개 취할 때의 검색 능력을 비교하였는데 이들 블럭 정합 알고리즘를 각각 16:1 FSA1, 16:1 FSA2, 16:1 FSA4로 언급하며 표 3에 이의 결과를 보인다. 표에 보인 것처럼 각 화소 그룹에 대해 후보 움직임 벡터수를 증가 시킬 수록 보다 정확한 움직임 벡터를 구하는 것을 알 수 있다. 그러나 16:1 FSA4와 16:1 FSA2의 성능 차이 보다는 16:1 FSA2와 16:1 FSA1의 성능차가 좀 더 큰 것을 알수 있다. 그리고 각 화소 그룹에 대해 후보 움직임 벡터의 수를 증가 시킬 수록, 후보 움직임 벡터의 수가 증가되므로 더욱 많은 계산량이 필요하다. 또한 16:1 FSA4를 구현할 경우 16:1 FSA2에 비하여 매우 많은 하드웨어가 필요한데 비하여, 16:1 FSA2와 16:1 FSA1을 구현하는데 필요한 하드웨어 차이가 크지 않아 16:1 FSA2를 본 논문에서 설계할 동작 예측 방법으로 결정하였다.

표 3. 후보 움직임 벡터 갯수에 따른 평균 PSNR.

Table 3. Average PSNRs according to the number of CMV(Candidate Motion Vector).

(Unit:dB)

Algorithm	Football	Table Tennis	Flower garden
16:1 FSA1	22.892	28.049	23.758
16:1 FSA2	22.940	28.135	23.831
16:1 FSA4	22.956	28.184	23.875

표 4. 여러 블럭 정합 알고리즘의 평균 PSNR.

Table 4. Average PSNRs of different block matching algorithms.

(Unit:dB)

Algorithm	Football	Table Tennis	Flower garden
16:1 FSA	22.940	28.135	23.831
FSA	22.968	28.224	23.923
4:1 FSA	22.951	28.183	23.893
TSS	22.172	26.292	21.734

이에 따라 다음 실험에서는 16:1 FSA에서 각 화소 그룹에 대해 2 개 후보 움직임 벡터를 선택하게 하고 기존의 블럭 정합 알고리즘과 그 성능을 비교하였다. 그림 4, 5, 6에 각각 'Football', 'Table tennis', 'Flower Garden' 영상에 대한 16:1 FSA, 4:1 FSA, FSA, TSS 등의 성능을 그리고 표 4에 각 영상에서 블럭 정합 알고리즘들의 평균 성능을 보인다.

이러한 실험 결과는 16:1 FSA의 성능이 TSS보다는 훨씬 우수하며, 4:1 FSA뿐만 아니라 FSA와도 유사함을 나타낸다. 이에 따라 16:1 FSA가 MPEG-I과 같은 응용에도 좋은 성능을 보이는지를 알아보기 위하여 다음과 같은 실험을 추가로 수행하였다.

실험은 현재 public domain에서 구할 수 있는 MPEG-I 부호화 프로그램인 PVRG_MPEG[4]에 시험할 동작 예측 알고리즘을 porting하여 여러 동영상 압축에 적용하였다. 동작 예측 알고리즘의 성능은 이러

한 실험으로 얻어지는 PSNR을 비교하여 그 우수성을 판정하였다. 블럭 크기는 MPEG-I에서 규정한 대로 16×16으로 하였으며 검색 영역의 크기는 47×47로 하여 telescopic 검색[6] 방법을 사용하였다.영상은 모두 I,B,B,P,B, B,I,B,B,P,B,B,I,... 순으로 부호화하며[6] 전송률은 1.5 Mbps로 하였다.

표 5에 각 영상에서 블럭 정합 알고리즘의 평균 성능을 보인다.이러한 실험 결과에 따라 16:1 FSA는 MPEG-I에 적용하여도 4:1 FSA뿐만 아니라 FSA와 유사한 성능을 가짐을 알 수 있다.

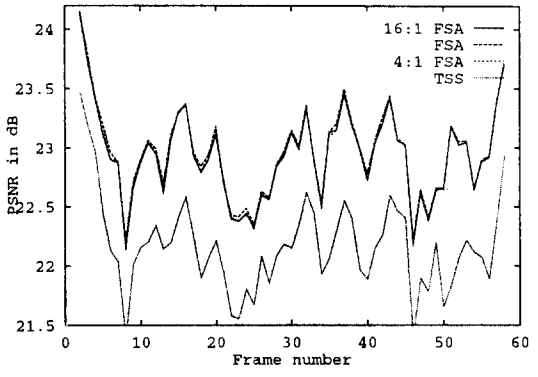

그림 4. Football 영상에서 여러 블럭 정합 알고리즘에 프레임단위의 PSNR비교.

Fig. 4. In football sequence comparison among different block matching algorithms in PSNR/frame.

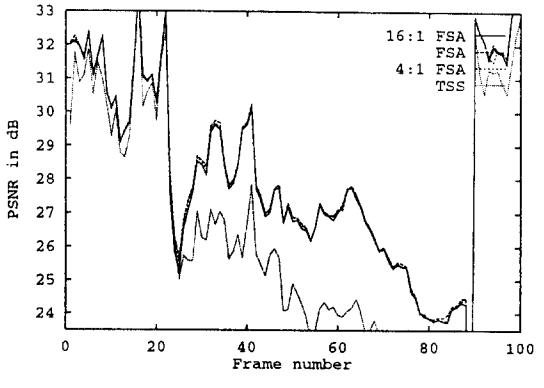

그림 5. Table tennis 영상에서 여러 블럭 정합 알고리즘간에 프레임단위의 PSNR비교.

Fig. 5. In table tennis sequence comparison among different block matching algorithms in PSNR/frame.

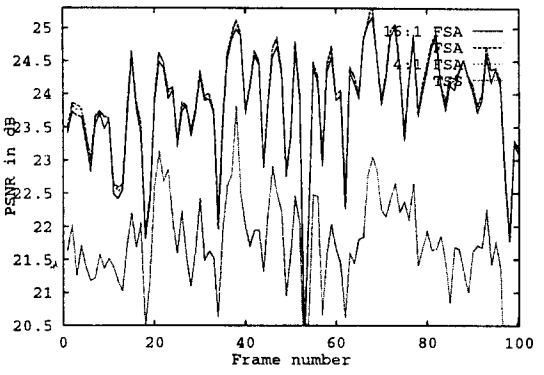

그림 6. Flower garden 영상에서 여러 블럭 정합 알고리즘간에 프레임단위의 PSNR비교.

Fig. 6. In flower garden comparison among different block matching algorithms in PSNR/frame.

표 5. 여러 블럭 정합 알고리즘의 평균 PSNR.

Table 5. Average PSNRs of different block matching algorithms.

(Unit:dB)

Algorithm	Football	Table Tennis	Flower garden
16:1 FSA	34.728	40.480	30.905
FSA	34.730	40.513	30.924
4:1 FSA	34.728	40.504	30.920
TSS	34.415	38.993	29.611

Ⅲ. 하드웨어 구조

이 장에서는 검색 영역과 블럭의 크기가 각각 47×47, 16×16 일때 16:1 FSA의 하드웨어 구조에 대해서 설명한다[14]. PMAD나 MAD는 두 화소의 차이의 합을 계산에 사용한 화소의 수로 나누어 계산되므로 편의상 이 장에서는 화소 그룹 또는 현재 영상 블럭과 검색 위치에서의 두 화소의 차이의 합으로 정의하며 이는 움직임 벡터 계산에 영향을 미치지 않는다.

그림 7에 전체 구조를 보인다. Input module은 현재 영상 블럭과 이에 대응하는 검색 영역에 대해 한 사이클에 4 개 화소의 자료를 동시에 받아 이를 processors array에 표 6에 따라 분배한다. Output module은 32 개의 후보 움직임 벡터와 이의 MAD를 저장하기 위한 메모리를 제공한다. Output module은 각 화소 그룹에 대해 PMAD를 최소로 하는 2 개의 후보 움직임 벡터(따라서 총 32 개의 후보 움직임 벡터)

와 이의 PMAD를 processors array로부터 받아 저장하고, 계속해서 processors array로부터 32 개 후보 움직임 벡터가 가리키는 검색 위치에서 16 개의 모든 화소 그룹에 대한 PMAD를 적절히 받아 이를 누적시켜 32 개의 MAD를 구하는 것이다. 그리고 최소의 MAD를 선택하며 이의 후보 움직임 벡터를 현재 영상 블럭에 대한 움직임 벡터로 선택된 MAD와 함께 출력한다.

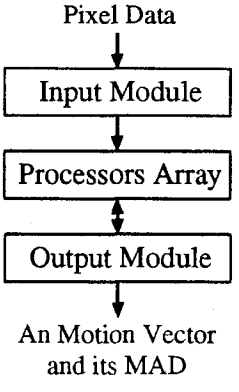

그림 7. 16:1 FSA의 전체 하드웨어 구조
Fig. 7. The overall structure.

Processors array는 원구조로 연결된 16 개의 동일한 프로세서로 구성된다. 하나의 프로세서는 임의의 검색 위치에서 PMAD 계산 능력을 가진다. 그림 8에 프로세서의 구성도를 보인다. 그림 8에 보인바와 같이 프로세서 i의 제어 신호는 레지스터를 거쳐 프로세서 i의 동작을 제어함과 동시에 프로세서 $i+1$로 전송된다. 따라서, 현재 클럭 사이클의 프로세서 i의 제어신호를 다음 클럭 사이클의 프로세서 $i+1$의 제어신호로 사용한다. 이것은 프로세서가 PMAD를 계산한 후, output module에 PMAD의 전송시기를 프로세서마다 다르게 한다. 그러므로 자연스러운 자료 흐름을 가지므로 전체 구조가 단순해 진다.

프로세서는 PMAD 계산에 사용되는 화소 그룹과 검색 영역의 화소를 저장하기 위해 각각 3 개, 2 개의 메모리 모듈을 가진다. 하나 이상의 메모리 모듈을 가지는 이유는 double buffering을 위해서이다. 즉, 프로세서가 일을 수행하는 동안 다음 움직임 벡터 계산에 필요한 화소 자료를 미리 받아 현재 움직임 벡터 계산에 이용되지 않는 메모리 모듈에 이를 저장함으로써 데이타 전송에 따른 지연을 없애기 위함이다.

프로세서가 움직임 벡터 계산을 위한 화소 자료를 모두 받았을때, 프로세서 i는 레이블 i가 부여된 검색 위치의 화소와 현재 영상 블럭에 대해 표 6에 보인 레이블이 부여된 화소를 모두 가진다. 이 화소를 이용하여 프로세서는 3 단계에 걸쳐 작업을 수행한다.

표 6. 화소 자료의 분배
Table 6. The Distribution of Pixel data.

Porcessor	Label of Search Location	Label of Pixel in Current Block
0	0	0
1	1	7
2	2	10
3	3	13
4	4	11
5	5	4
6	6	1
7	7	14
8	8	2
9	9	5
10	10	8
11	11	15
12	12	9
13	13	6
14	14	3
15	15	12

첫 단계에서 프로세서 i는 프로세서가 가진 화소 그룹의 레이블과 같은 레이블을 가지는 모든 검색 위치에서의 PMAD를 계산하여 각 화소 그룹에 대한 2 개의 후보 움직임 벡터를 선택한다.

예로, 프로세서 6은 검색 위치의 레이블이 6인 화소와 화소 그룹 P_1의 화소를 가진다. 이들 화소를 이용해 프로세서 6은 레이블이 1인 모든 검색 위치의 $PMAD_1$를 계산한다(그림 9 (a) 참조). 각 프로세서가 계산한 PMAD의 검색 영역 레이블은 표 7의 첫 행에 보인다. 프로세서는 검색 영역에서 각 검색 위치의 PMAD를 raster scan 방식과 같은 차례로 계산한다. 이 동안 프로세서는 가장 작은 2 개의 PMAD와 이의 후보 움직임 벡터를 유지하여 첫 단계의 종결시에는 가장 작은 2 개의 PMAD와 이의 후보 움직임 벡터를 얻을 수 있다. 그러므로 각 프로세서가 첫 단계에서 수행하는 일은 가장 작은 PMAD 2 개를 유지하는 것을 제외하고 검색 영역과 블럭의 사이즈가 각각 11×11, 4×4일때 FSA를 수행하는 것과 동일하다.

PMAD 계산시 가장 어려운 것은 메모리 주소 발생이다. 16:1 부분 표본 추출된 검색 영역중 레이블 k를 가지는 검색 위치를 부분 검색 영역 k(혹은 단순히 부

분 검색 영역)라고 하자. 만일 부분 검색 영역 k에서 좌표가 (x,y)이면, 실제 검색 영역에서 이에 해당되는 좌표는 $(4x+a, 4y+b)$로 표현할 수 있다. a와 b의 값은 검색 위치에 부여된 레이블 k에 의해 결정된다. k를 $(i_3i_2i_1i_0)_2$로 2진수로 나타내면, a와 b는 각각 $(i_1i_0)_2$과 $(i_3i_2)_2$에 해당된다. (x,y)와 (a,b) 두 순서쌍을 base 후보 움직임 벡터, offset 후보 움직임 벡터로 각각 언급하며 base 후보 움직임 벡터와 offset 후보 움직임 벡터를 이용하여 프로세서는 부분 검색 영역을 저장한 메모리의 주소를 발생한다.

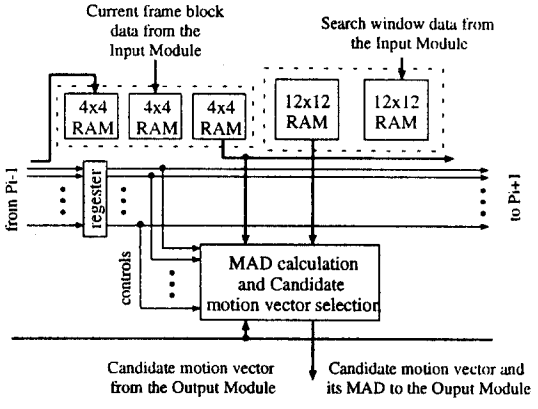

그림 8. 프로세서의 구성도
Fig. 8. The processor organization.

그림 9 (a)와 (b)에 각각 프로세서 6과 14에서의 $PMAD_1$, $PMAD_3$ 계산 예를 보인다. 그림에서 굵고 가는 선들로 둘러쌓인 정사각형은 각각 검색 영역와 현재 영상 블럭의 화소를 나타내며, 각 사각형내의 왼쪽 상단과 오른쪽 하단 구석의 숫자는 각각 검색 영역의 검색 위치와 현재 영상 블럭의 화소 레이블을 나타낸다.

그림 9의 두 예는 모두 검색 위치가 base 후보 움직임 벡터단위로(따라서 부분 검색 영역의 좌표계로) (0,0)일때이다. 그러나 그림에서 부분 검색 영역의 화소 좌표가 서로 다르다. 프로세서 i가 검색 위치 $(4x+a, 4y+b)$에서의 PMAD를 계산해야 할 경우를 생각해 보자. i를 $(i_3i_2i_1i_0)_2$로 2진수 표현하고 $c=(i_1i_0)_2$, $d=(i_3i_2)_2$라 하자. 그러면 PMAD 계산을 위한 부분 검색 영역에서 화소 좌표 $(\tilde{x}, \tilde{y})$를 다음의 식에 의해 계산한다.

$$\tilde{x}=\begin{cases} x+1 & \text{if } a>c \\ x & \text{otherwise,} \end{cases} \qquad \tilde{y}=\begin{cases} y+1 & \text{if } b>d \\ y & \text{otherwise} \end{cases}$$

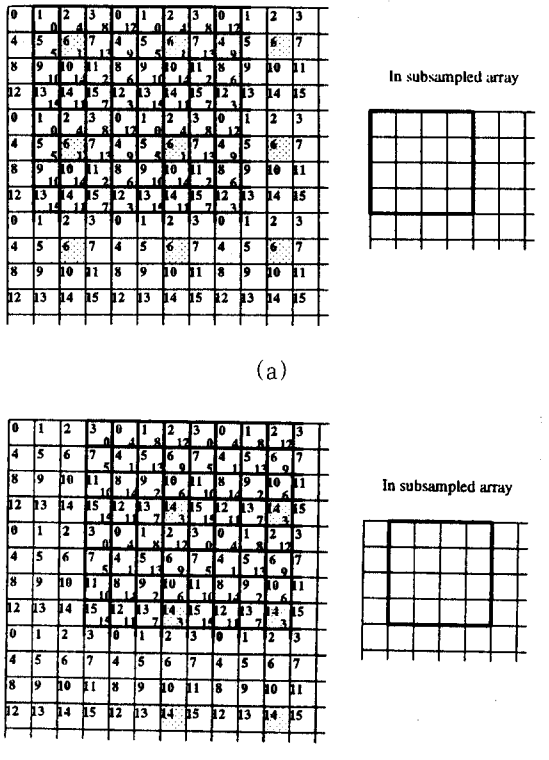

그림 9. PMAD 계산의 예
Fig. 9. Examples of PMAD calculations.

두 번째 단계에서 각 프로세서는 base 후보 움직임 벡터와 offset 후보 움직임 벡터의 두 쌍과 이의 PMAD를 output module에 round robin 방식으로 전송한다. Output module은 이들을 자신의 메모리에 offset 후보 움직임 벡터를 주소로 하여 저장한다. 이와 동시에 프로세서 i(15인 경우) ($=0,1,\cdots,14$)는 프로세서 $i+1$(0에 해당)에 자신이 현재 저장중인 화소 그룹을 세 번째 단계를 위해 전달한다. 이로 인해 프로세서는 현재 영상 블럭에 대해 3 개의 메모리 모듈이 필요하다. 2 개 메모리 모듈은 앞에서 언급했듯이, 외부로부터 입력 받은 화소 자료를 저장하기 위하여, 그리고 나머지 모듈은 화소 자료를 프로세서간에 순환하기 위하여 사용된다.

화소 그룹을 순환시키는 이유는 각 후보 움직임 벡터가 가리키는 검색 위치에서 현재 영상 블럭의 모든 화소를 이용해서 MAD를 계산하기 위함이다. 한 예로, 프로세서 6은 첫 번째 단계에서 $PMAD_1$을 계산한다. 두 번째 단계가 종결후 프로세서 6은 화소 그룹 P_4를 가지며, 세 번째 단계에서 프로세서 9가 첫 단계에서

구한 2 개의 후보 움직임 벡터가 각각 가리키는 검색 위치의 $PMAD_4$를 계산한다. 같은 검색 위치에서 PMAD계산에 임의의 두 화소 그룹을 사용하더라도, 이에 연관되어 사용된 화소들의 집합은 공유되지 않는 특성을 16:1 FSA는 가진다. 따라서, MAD는 16 개 화소 그룹에 대한 PMAD를 모두 더해야 계산되므로, 32 개 MAD를 구하기 위해 화소 그룹 순환을 16 번 하며 화소 그룹 순환마다 하나의 프로세서는 2 개의 PMAD를 구한다. 표 7에 화소 그룹의 순환 후에 각 프로세서가 계산하는 PMAD의 검색 위치 레이블을 보이며 0 행은 첫 단계에서 검색 위치의 레이블을 보인다. 각 프로세서들은 자신에 해당되는(표 7에서 프로세서 i는 자신에 해당되는 i 번째 열의 내용) 검색 위치의 레이블을 롬(ROM) 테이블로 가지고 있으며 각 검색 위치의 PMAD의 계산시에 자신의 롬 테이블을 읽어 각 검색 위치의 레이블을 알게 된다.

표 7. 각 화소 그룹 순환후의 프로세서가 계산하는 PMAD의 검색 영역 레이블.
Table 7. Labels of the search locations where the processors calculate the PMADs after each rotation.

processor id / rotation	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	7	10	13	11	4	1	14	2	5	8	15	9	6	3	12
1	1	1	4	11	2	8	5	2	3	3	6	9	0	10	7	0
2	5	2	2	5	12	3	9	6	7	0	0	7	14	1	11	4
3	9	6	3	3	10	13	0	10	11	4	1	1	8	15	2	8
4	13	10	7	0	4	11	14	1	15	8	5	2	6	9	12	3
5	4	14	11	4	5	5	8	15	6	12	9	6	7	7	10	13
6	2	5	15	8	9	6	6	9	0	7	13	10	11	4	4	11
7	12	3	6	12	13	10	7	7	14	1	4	14	15	8	5	5
8	10	13	0	7	1	14	11	4	8	15	2	5	3	12	9	6
9	11	11	14	1	8	2	15	8	9	9	12	3	10	0	13	10
10	15	8	8	15	6	9	3	12	13	10	10	13	4	11	1	14
11	3	12	9	9	0	7	10	0	1	14	11	11	2	5	8	2
12	7	0	13	10	14	1	4	11	5	2	15	8	12	3	6	9
13	14	4	1	14	15	15	2	5	12	6	3	12	13	13	0	7
14	8	15	5	2	3	12	12	3	10	13	7	0	1	14	14	1
15	6	9	12	6	7	0	13	13	4	11	14	4	5	2	15	15

화소 그룹의 순환후의 세 번째 단계의 과정은 다음과 같다. 먼저, 프로세서는 output module에게 표 7에 보인 offset 후보 움직임 벡터를 보낸다. 그러면 output module은 offset 후보 움직임 벡터를 주소로 하여 자신의 메모리로부터 base 후보 움직임 벡터를 읽어 이에 연관된 프로세서(offset 후보 움직임 벡터를 보낸 프로세서)에게 보낸다. 프로세서는 base 후보 움직임 벡터를 받아 PMAD 계산을 시작하며, 계산한

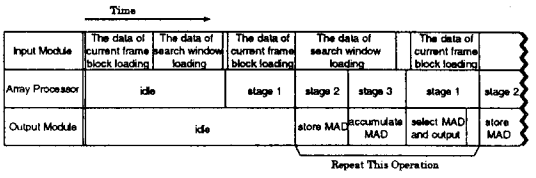

그림 10. 전체 동작도
Fig. 10. The overall operation.

PMAD를 MAD 계산을 위해 output module에게 보낸다. 각 화소 그룹에 대해 2 개의 후보 움직임 벡터가 존재하며 따라서 프로세서는 하나의 화소그룹에 대해 2 개의 PMAD를 계산해야 한다. 이에 따라 화소 그룹의 2 번째 PMAD를 계산시에 화소 그룹의 프로세서간에 순환이 일어난다. 이러한 과정을 통해 각 프로세서가 30개의 PMAD를 계산하여 output module에 전송하면, output module은 32 개의 MAD를 얻으며 세 번째 단계가 종료된다.

그림 10에 전체 동작을 보인다. Input module은 먼저 현재 영상 블럭의 화소를 받은 후, 검색 영역의 화소를 받는다. 두 번째, 세 번째 단계에서 화소 그룹의 순환이 발생하기 때문에 input module은 두 번째 단계가 시작하기 전에 현재 영상 블럭을 받아야 한다. 만일 input module이 두 번째 단계전에 현재 영상 블럭의 화소를 모두 받지 못하면, 두 번째 단계는 input module이 현재 영상 블럭을 모두 받을때 까지 연기된다. 이와 유사하게 입력 모듈이 세 번째 단계 종료전까지 검색 영역의 화소를 모두 받지 못하면, 다음 블럭의 첫 단계는 검색 영역의 화소를 모두 받을때까지 연기된다.

지금까지 설명한 동작 예측기를 삼성의 KG60000 sea-of-gates 라이브러리를 이용하여 게이트 레벨로 설계하였다. 설계한 동작 예측기는 nand 게이트 기준으로 총 41,514 개의 게이트와 43,714 bit의 RAM과 80 bit의 ROM을 사용하였다. 검색 영역와 현재 영상 블럭의 크기는 각각 47×47과 16×16로 하였고, VHDL을 이용해 회로를 묘사하고, 이의 작동을 검증하였다.

본 논문에서 제안한 동작 예측기는 1,674 클럭 사이클마다 하나의 움직임 벡터와 이의 MAD를 출력한다. MPEG-I 부호기는 크기가 352×288인 영상을 초당 25 개 영상을 처리할 것을 요구하는데[6], 이는 초당 19,800 개의 움직임 벡터 계산량에 해당되며 33 MHz의 클럭 동작에서 1,684 클럭마다 하나의 움직임 벡터

를 계산을 요구한다. 따라서 제안한 동작 예측기를 구현하여 33 MHz의 클럭 스피드로 작동시키면 MPEG-I 부호기에 사용될 수 있다. 참고로 이는 MPEG-I 부호기가 telescopic 검색을 할 때로 가정한 경우이다.

Ⅳ. 결 론

본 논문에서는 16:1 FSA라는 새롭고 빠른 블럭 정합 알고리즘와 이의 하드웨어 구조을 제시하였다. 이 블럭 정합 알고리즘는 Liu and Zaccalin[8]에 의해 제시된 4:1 FSA의 알고리즘을 모태로 한 교대적인 16:1 부분 표본 추출 방법을 사용한 것이다. 이 방법은 계산량이 4:1 FSA에 비해 약 4분의 1로 감소하였지만, 성능은 4:1 FSA 뿐만 아니라 FSA와도 유사하다. 이의 하드웨어는 nand 게이트를 기준으로 약 42,000 개의 게이트와 43,000 비트의 메모리를 사용하며 이는 단일 IC chip으로 구현이 가능한 수준이다. 이러한 동작 예측기를 구현하여 33 MHz 클럭 스피드로 작동시키면 실시간용 MPEG-I 부호기의 동작 예측에 사용될 수 있다. 앞으로 16:1 FSA의 최대 장점인 필요한 계산이 적으면서도 우수한 동작 예측 결과를 낸다는 점을 살려, MPEG-II 그리고 HDTV에 적용 가능 여부를 연구할 필요가 있다.

참 고 문 헌

[1] M. Bierling, "Displacement estimation by hierarchical blockmatching," in Proc. SPIE Conf. Visual Communications and Image Processing '88, Cambridge, MA, Nov. 1988, vol. 1001, pp. 942-951, 1988.

[2] L. G. Chen, W. T. Chen, Y. S. Jehng and T. D. Chiueh, "An efficient parallel motion estimation algorithm for digital image processing," IEEE Trans. on Circuits and Systems for Video Technology, Vol. 1, No. 4, pp. 378-385, December 1991.

[3] C. Gonzales and E. Viscito, "Flexibly scalable digital video coding," Signal Processing:Image Communcation, Vol. 5, No. 1-2, pp. 5-20, February 1993.

[4] A.C.Hung, "PVRG-MPEG CODEC 1.1," Stanford University, May 1993.

[5] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," IEEE Trans. on Commun., Vol. COM-29, pp. 1799-1808, December 1981.

[6] ISO/IEC/JTC1/SC29/WG11 Systems Committee, "Coding of moving pictures and associated audio," March 1992.

[7] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," IEEE Trans. on Circuits and Systems, Vol. 36, No. 10, pp. 1301-1308, October 1989.

[8] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," IEEE Trans. on Circuits and Systems for Video Technology, Vol. 3, No. 2, pp. 148-157, April 1993.

[9] G. Morrison and I. Parke, "A compatible scheme for moving image coding," Signal Processing:Image Communcation, Vol. 5, No. 1-2, pp. 91-103, February 1993.

[10] H. G. Musmann, P. Pirsch, and H. J. Gralleer, "Advances in picture coding," Proc. IEEE, vol. 73, no. 4, pp. 523-548, April 1985.

[11] R. Srinivasan and K. R. Rao, "Predictive coding based on efficient motion estimation," IEEE Trans. on Commun., Vol. COM-33, pp. 1011-1014, September 1985.

[12] K. M. Yang, M. T Sun and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," IEEE Trans. on Circuits and Systems, Vol. 36, No. 10, pp. 1317-1325, October 1989.

[13] L. D. Vos and M.Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," IEEE Trans. on Circuits and Systems, Vol. 36, No. 10, pp. 1309-1316, October 1989.

[14] 임종석 외 4인, 동화상 압축/복원용 동작 예측기 구조에 관한 연구, 수탁과제 연구보고서, 한국전자통신연구소, 1994

저 자 소 개

金 良 勳(正會員)

1993年 2月 서강대학교 전자계산학과 학사. 1995年 2月 서강대학교 전자계산학과 석사. 1995年 3月 ~ 현재 LG전자 정보시스템 연구소 PC 연구실 연구원. 주관심분야는 동화상 압축, ASIC 설계 등임

閔 丙 基(正會員)

1980년 서울대학교 전자공학과 학사. 1982년 한국과학기술원 전기 및 전자공학과 석사. 1991년 프랑스 고등통신원(ENST) 박사. 1982년 ~ 현재 한국전자통신연구소 미디어 연구실 책임연구원. 주관심분야는 VLSI 구조, 영상 및 신호처리, ASIC 설계 등임.

林 鍾 錫(正會員)

1981年 서강대학교 전자공학과 학사. 1983年 한국과학기술원 전기 및 전자공학과 석사. 1989年 Univ. of Maryland, College Park, 전기공학과 박사. 1983年 3月 ~ 1990年 8月 한국 전자통신연구소 연구원. 1990年 9月 ~ 현재 서강대학교 전자계산학과 부교수.