삼단검색 알고리즘을 위한 움직임 추정기 구조

김 상 중[†] · 김 용 길[††] · 임 강 빈[††] · 김 용 득[††] · 정 기 현[††]

요 약

본 논문에서는 동영상 처리에서 사용되는 움직임 추정을 위한 삼단 검색 알고리즘의 구조를 제안한다. 제안된 구조에서는 규칙적인 데이타 입력이 가능하게 되고 입력된 데이타는 그 데이타를 사용하는 모든 연산과정을 거치게 되어 데이타 입력 대역폭이 최소화된다. 기존의 구조들보다 적은 자원으로 최대 계산 속도에 접근하는 성능을 가진다. 제안된 구조의 성능 분석과 다른 구조들과의 비교가 제시된다.

A High Speed Motion Estimation Architecture for Three Step Search Algorithm

Sang Joong Kim[†] · Yong Gil Kim[††] · Kang Bin Yim[††] · Yong Deak Kim[††] · Ki Hyun Chung[††]

ABSTRACT

We propose a new VLSI architecture for the three step search algorithm for the motion estimation of moving images. In the proposed architecture the regular data input is possible and the data are passed through all computational processes, minimizing the input bandwidth. The performance is analyzed in detail, and compared with other architectures. The performance is approaching to the ideal computation speed, with less hardware than for the existing architectures.

1. 서 론

움직임 추정 알고리즘은 크게 블럭 정합(Block Matching)과 화소 순환(Pel Recursive)방식으로 나눌 수 있다. 이 두 방법을 사용하면 움직임 벡터를 구하기 위해서는 많은 계산량이 요구된다. 블럭 정합 방식은 화소 순환 방식보다는 적은 계산량이 요구되고 구조의 규칙성이 뛰어나 하드웨어 구현이 용이하므로 많이 사용되고 있다.

† 정 회 원: 한국전자통신연구원
†† 정 회 원: 아주대학교 전자공학과
논문접수: 1996년 10월 14일, 심사완료: 1997년 2월 3일

블럭 정합 방식중 완전 탐색 블럭 정합 알고리즘(Full Search Block Matching Algorithm)[1]이 가장 널리 쓰이는 방법이다. 이 방식은 다른 구조에 비하여 움직임 벡터 추정을 위해 많은 계산량이 요구되는 단점을 지니지만 연산 구조가 간단하고 규칙적인 장점이 있다. 이 알고리즘을 이용하여 SIMD-Systolic Array 구조[2]로 구현한 경우 SIMD시스템과 시스톨릭 어레이의 장점을 채택하고 있으나 높은 입력 대역폭이 요구되며 병렬처리나 파이프라인 구조가 아니므로 오버헤드가 많다. 다른 방식으로는 삼단 검색(Three-Step Search)알고리즘[3]이 있다. 삼단 검색 알고리즘은 완전 탐색 블럭 정합 알고리즘에 비해 요구되는 계산량이 비교적 적다는 장점이 있다. 움직임 추정 알고리

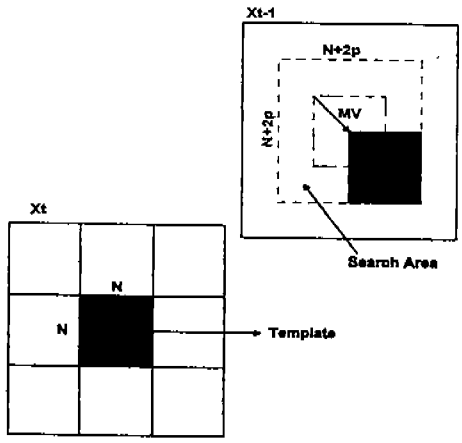
즘은 계산의 복잡성, 데이터의 규칙성, 구현시 가격으로 비교할 수 있다. 완전 탐색 정합 알고리즘은 데이터의 규칙성은 있으나 많은 계산량과 시간이 필요하게 되고 실현 가격 면에서도 비교적 불리하다. 이에 반해, 삼단 검색 알고리즘은 적은 계산량을 요구하므로 짧은 시간내에 움직임 추정 벡터를 구할 수 있으나, 구현상 몇가지 어려움이 있다. 첫째는 전단계의 계산이 완전히 끝난 후에 다음 단계를 수행할 수 있으므로 파이프라인이나 병렬처리가 어려우며, 둘째로는 각 단계별로 탐색영역의 크기가 다르므로 입력되는 데이터의 규칙성이 없다는 점이다.

본 논문에서는 삼단 검색 방법을 이용한 새로운 움직임 추정기의 구조를 제안한다. 제안된 구조에서는 각 단계에서 입력되는 데이터를 최대한 규칙적으로 입력하게 하여 입출력 데이터의 대역폭을 최소화하고, 최소의 자원으로 이를 효과적으로 수행하기 위한 고속 움직임 추정기를 제안한다. 본 논문의 구성은 다음과 같다. 제2절에서는 블럭정합을 위한 알고리즘들을 소개하고 각 알고리즘의 장단점을 살펴본다. 제3절에서는 삼단 검색 방법을 이용한 고속 움직임 추정기의 구조를 제안하며 이 구조의 동작원리를 설명하고 다른 구조와 성능 비교를 한다. 끝으로 제4절에서 결론을 제시한다.

2. 블럭 정합에 의한 움직임 추정

블럭 정합 방식은 한 영상을 서로 겹치지 않는 여러 개의 작은 기준 블럭(Template)들로 나누고, 모든 기준 블럭을 이전 영상내 대응되는 참조 블럭(Candidate)의 주변 영상과 서로 비교하고 미리 정해진 크기의 탐색 영역내에서 블럭을 반복적으로 비교하며 가장 잘 일치하는 참조 블럭의 변위를 해당 기준블럭의 움직임 벡터로 취하는 방법이다.

(그림 1)은 동영상에서 시간축 상에서 이웃한 프레임 X_t와 X_{t-1}사이에 움직임이 있을 경우 움직인 물체내의 화소들은 같은 움직임 벡터를 가지며, 이러한 움직임이 있는 화소들로 구성된 작은 크기의 블럭은 같은 움직임 벡터를 가지고 있다고 가정한다. 이런 가정을 바탕으로 블럭 정합 알고리즘에서는 현재의 영상은 N×N 크기의 기준 블럭들로 나누어진다. 이런 각각의 블럭들은 이전 영상의 탐색영역내에서 가장 잘 정합되는 블럭에서 움직임이 있었다고 가정하고 이때 움직인 변위를 움직임 벡터라 한다. 움직임 벡터는 (그림 1)에서와 같이 기준 블럭과 가장 잘 정합되는 탐색 영역내의 블럭간의 변위가 된다. 탐색영역에서 가장 잘 정합되는 블럭을 찾기위하여 기준블럭을 상하좌우로 이동한다. 이때 최대 이동 변위를 p라 하면, 탐색 영역은 (그림 1)에서와 같이 $(N+2p)^2$개의 화소들로 구성되므로 전체 정합을 해야하는 블럭의 수는 $(2p+1)^2$ 개가 된다.



(그림 1) 기준블럭과 탐색영역에서의 움직임 벡터
(Fig. 1) The template block and motion vector of a search area

블럭정합 알고리즘을 이용하여 구해진 움직임 추정 벡터의 값은 실제 화면상의 물체의 운동벡터와 일치하지 않고 단지 두 블럭 사이의 왜곡이 최소가 되는 블럭의 운동 벡터이다. 왜곡의 계산에는 오차거리 함수(Error Distance Function)가 많이 사용된다. 오차거리 함수로는 SAD(Sum of Absolute Difference)를 많이 사용하는데, 이는 곱셈연산 없이 덧셈연산 만으로 수행되므로 하드웨어 구현이 쉽고 비교적 좋은 성능을 보이기 때문이다.

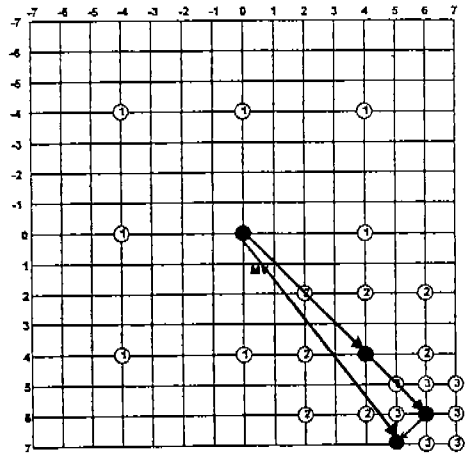
완전 탐색 정합 알고리즘에서 한 개의 기준 블럭에 대한 움직임 벡터를 계산하기 위해서는 $N^2\times(2p+1)^2$의 산술 연산이 요구된다. 초당 필요한 계산량을 산정하기 위해 H.261에서 표준 영상[5]으로 예를 들면 하나의 CIF(Common Intermediate Format)영상은 352×288의 해상도를 가지므로 16×16크기의 22×18개의

블럭으로 구성된다. 이동량의 최대변위는 8이고 초당 30 프레임을 처리해야하는 경우 실시간 처리를 위한 전체 연산량은 약 780 mops(million operation per seconds)가 요구된다.

완전 탐색 정합 알고리즘의 과다한 계산량을 피하기 위한 방법으로는 삼단검색 알고리즘을 사용하며 이 알고리즘은 세 단계로 움직임 벡터를 계산한다. 즉, 알고리즘의 첫번째 단계는 간격이 비교적 멀리 떨어져 있는 여러 개의 예상 벡터들을 이용하여 근사적인 움직임 벡터를 검색한다. 이렇게 하여 정해진 움직임 벡터를 원점으로 하여 두번째 단계의 움직임 벡터를 찾는다. 이 경우 예상 벡터들의 위치는 첫번째 단계에서 보다 적은 간격을 갖게 된다. 마지막 단계는 두번째 단계에서 얻어진 움직임 벡터를 원점으로 하여 더욱 간격이 적어진 예상 벡터들을 대상으로 최종적인 움직임 벡터를 찾는다.

(그림 2)는 삼단검색 알고리즘의 한 예를 나타낸 것이다. 첫번째 단계에서는 (0, 0)을 중심으로 간격이 4인 9개의 참조블럭을 정합하여 그중 최소의 왜곡이 있는 점을 찾는다. 두번째 단계에서는 첫번째 단계에서 찾은 점을 중심으로 간격이 2인 9개의 블럭을 정합하여 최소의 왜곡을 찾는다. 세번째 단계에서는 두번째 단계에서 찾은 점을 중심으로 간격이 1인 9개의 블럭을 정합하여 왜곡이 최소인 점을 움직임 벡터로 정한다. 이 알고리즘의 가장 큰 장점은 블럭 정합하는 횟수를 감소시킬 수 있다는 것이다. CIF영상을 처리하는데 필요한 계산량의 예를 들어 보자. 만일 기준 블럭을 16×16으로 나누면 22×18개의 블럭이 생기고 초당 30 프레임을 처리해야 하므로 실시간 처리를 위해서는 약 82 mops (3×9×22×18×30×16×16 =82,114,560)가 필요하다.

앞에서 언급한 완전 탐색 정합 알고리즘 (780 mops)에 비하면 매우 적은 계산량으로 움직임 추정을 할 수 있음을 알 수 있다. 그러나, 이 알고리즘은 몇 가지 단점을 갖고 있다. 첫째, 앞 단계의 계산이 완전히 끝난후에 다음 단계의 계산이 가능하므로 완전 탐색 정합 알고리즘 구조에 많이 이용되는 병렬처리가 어려우며 둘째로는 각 단계의 검색영역의 크기가 다르므로 다른 알고리즘에 비해서 규칙성이 낮아 하드웨어로 구현하기가 어렵다는 점이다. 또하나의 단점은 완전 탐색 정합 알고리즘을 사용할 경우보다 복원된 영상의 화질이 떨어질 수 있다는 점이다. 그러나 삼단검색 알고리즘으로 복원된 동영상의 화질은 완전탐색 알고리즘에 의해 얻어진 그것과 큰 차이가 없는 것으로 알려져 있다. 따라서 앞에서 열거한 두 문제점을 해결하면 삼단검색 알고리즘은 블럭 정합을 위해 널리 사용될 수 있을 것이다. 다음 절에서는 위의 문제점들을 해결할 수 있는 구조를 제안하고 성능을 분석한다.



(그림 2) 삼단 검색 알고리즘
(Fig. 2) The three step search algorithm

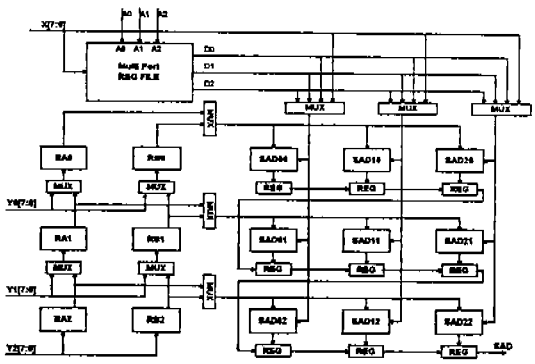
3. 삼단 검색 알고리즘을 이용한 고속 움직임 추정기

제안하는 추정기에서는 입력 데이터의 대역폭을 최소화하기 위하여 모든 데이터는 한 스텝당 한번만 입력되도록 한다. (그림 1)에서와 같이 (N +2p)×(N + 2p) 크기의 검색영역내에서 움직임 벡터를 구하기 위해서는 27×N×N의 데이타입력이 요구되며 완전 탐색 정합 알고리즘에서는 $(2p+1)^2 \times N \times N$의 데이터 입력이 요구된다. 참조영역의 데이타 입력포트가 R 개일때 삼단검색 알고리즘의 이론상 최대 처리 속도는 데이터를 한번 입력하는 데 걸리는 시간, 즉 (27×N×N)/R 사이클이 요구된다. 본 연구에서는 이 최대 처리 속도에 근접하는 구조를 제안하고자 한다.

(그림 3)은 본 연구에서 제안하는 구조이며 참조블

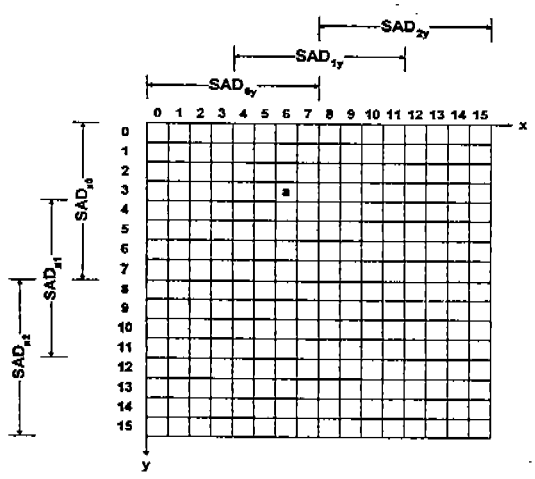
력의 입력을 위한 3개 포트를 갖는 SAD Unit의 블럭도이다. 여기서는 기준 블럭의 크기는 8×8 이고 검색영역은 16×16으로 가정한다. 각 단계에는 한 개의 기준블럭과 9개의 참조블럭이 정합을 하므로 9개의 프로세스 엘리먼트 SAD_{xy}로 구성되어 있다. 이때 $0 \le x \le 2$, $0 \le y \le 2$ 이다. 각 프로세스 엘리먼트는 3개씩 그룹이 지워져 있으며 같은 열에 있는 3개의 SAD는 참조블럭이 입력되는 데이터 버스를 공유한다. X[7:0]는 기준 블럭의 데이터를 입력하는 버스이며, Y0[7:0], Y1[7:0], Y2[7:0]는 참조 블럭의 데이터를 입력하는 버스이다. 탐색영역에서 참조블럭은 3개의 행으로 나누어 지고 각 행마다 3개의 참조블럭이 있다. 같은 행에 있는 참조블럭의 데이타는 3개의 버스 Y0[7:0], Y1[7:0], Y2[7:0]를 통하여 입력되며 각 참조블럭 사이에서 중첩이 되는 데이타는 쉬프트 동작을 사용하여 그 데이타를 요구하는 프로세서 엘리먼트에 적절히 공급하게 된다. X[7:0]와 연결되어 있는 레지스터 파일은 메모리로부터 입력되는 기준블럭의 데이터를 저장하는 동시에 각 프로세서 엘리먼트에 기준블럭의 데이타를 공급한다. 다음 단계부터 각 프로세서 엘리먼트는 기준 블럭 데이터를 레지스터 파일로부터 공급받는다. 이 레지스터 파일은 멀티포트 구조를 가지고 있으며 같은 열에 있는 프로세스 엘리먼트들은 같은 기준블럭의 데이타를 공유한다. (그림 3)에서 참조 블럭의 데이터를 적당한 프로세스 엘리먼트에 공급하기 위하여 6개의 레지스터를 사용한다. 이 레지스터들을 RA_x와 RB_x인 2개의 그룹으로 나누었다. 이때 $0 \le x \le 1$이다. 두 그룹의 레지스터는 교대로 각 프로세스 엘리먼트와 데이터 버스를 사용한다. 즉, 한 레지스터 그룹이 데이터를 입력받는 동안 다른 레지스터 그룹은 프로세스 엘리먼트에 데이터를 공급해 주어 계산속도를 증가시킨다. 또한 이 레지스터들은 입력단의 MUX를 통하여 버스의 데이터를 입력받거나 데이터 쉬프트 동작을 하게된다. 각 SAD출력단에는 레지스터가 연결되어 있으며 이 레지스터들은 SAD 계산한 결과를 저장하고 SAD 계산이 끝난 후 쉬프트 동작을 하여 움직임 벡터를 계산하는 블럭으로 데이터를 전송하게 된다.

(그림 4)는 삼단검색 알고리즘의 첫번째 단계에서 비교 연산시 필요한 검색영역을 나타낸다. 총 9개의 참조블럭으로 나누어지며 각 화소에 따라 기존 블럭

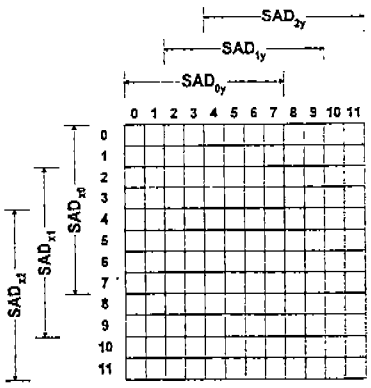


(그림 3) SAD Unit 블럭도(3개의 참조블록 입력포트)
(Fig. 3) The block diagram of SAD unit with 3 input ports for candidate blocks

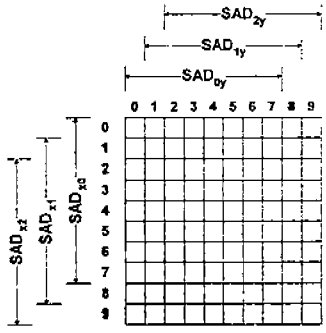
과 비교연산이 수행되는 SAD를 나타내고 있다. 예를 들어 (그림 4)에서 좌표 (6, 3) 위치에 있는 a점은 가로축 x에서는 SAD_{0x}, SAD_{1x}가 비교연산을 해야하며, 세로축 y에서는 SAD_{y0}가 비교연산을 해야한다. 그러므로 a점이 SAD에 입력이 되었을때 SAD_{00}, SAD_{10}에서 기준블럭과 비교연산이 수행된다. 여기서 $0 \le x \le 2$, $0 \le y \le 2$이며 가로축과 세로축의 값이 서로 대입이 된다. 예를 들면 세로축이 SAD_{x2}이고 가로축이 SAD_{0y}이면 x에는 0, y에는 2가 각각 서로 대입이 되어 SAD_{02}가 연산하는 영역을 표시하게된다. (그림 5)



(그림 4) 1 단계에서 데이터의 위치에 따라 연산하는 SAD
(Fig. 4) The data in a search area needed for the computation of each SAD at step 1



(그림 5) 2 단계에서 데이터의 위치에 따라 연산하는 SAD
(Fig. 5) The data in a search area needed for the computation of each SAD at step 2



(그림 6) 3 단계에서 데이터의 위치에 따라 연산하는 SAD
(Fig. 6) The data in a search area needed for the computation of each SAD at step 3

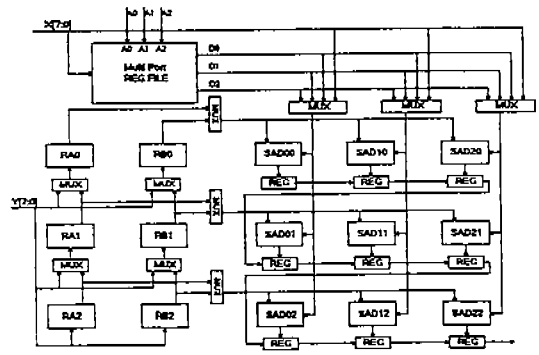
와 (그림 6)은 각각 2단계와 3단계에서 비교 연산시 필요한 검색영역과 각 화소의 위치에 따라 비교연산이 수행되는 SAD를 나타내고 있다.

(그림 7)은 본 연구에서 제안하는 참조블럭의 입력을 위한 1개 포트를 갖는 SAD Unit의 블럭도이다. (그림 3)의 SAD는 데이타입력을 위한 핀수가 많이 필요하지만 (그림 7)의 SAD는 작은 핀수를 가지고 같은 기능을 수행하는 구조이다. (그림 3)의 구조에서 Y0[7:0], Y1[7:0], Y2[7:0]를 (그림 7)의 구조에서 한 개의 Y[7:0] 버스로 묶고 병렬로 입력되는 데이타는 순차적으로 입력하게 된다. 그러므로 (그림 7)의 구조는 (그림 3)의 구조에 비하여 속도면에서는 불리한 반면 입력에 필요한 핀수에서는 유리한 점을 지니고 있다. 또한 〈표 1〉, 〈표 2〉, 〈표 3〉을 살펴보면 (그림 3)에서 3개의 참조블럭 입력 포트가 매단계마다 전부 사용되지는 않고 1개의 포트만 사용하는 경우가 많기 때문에 속도면에서의 (그림 7)의 구조보다 성능이 크게 향상되는 것은 아니다.

〈표 1〉, 〈표 2〉, 〈표 3〉은 (그림 3)의 구조를 사용할 때 데이터의 흐름을 표시하고 있다. 여기서 i는 참조블럭의 열 좌표이고 1단계에서는 0에서 15까지, 2단계에서는 0에서 11까지, 3 단계는 0에서 9까지 각각 증가하게 된다. 모든 좌표는 (0, 0)을 기준으로 각 참조블럭의 데이터가 갖는 오프셋(Offset)을 나타낸다. 또한 각 레지스터의 값이 갱신 될때 변하는 레지스터 값과 각 레지스터들이 동작하고 있는 상태를 L(x) (Load RA_x or Load RB_x), S(Shift), C(Calculate SAD)로 표기를 하였다.

〈표 4〉, 〈표 5〉, 〈표 6〉는 (그림 7)의 구조를 사용할 때 데이터의 흐름을 표시 하였다. 〈표 1〉, 〈표 2〉, 〈표 3〉에서 Y0, Y1, Y2 3개의 버스에 병렬로 입력되는 데이터를 한개의 버스를 사용하여 순차적으로 입력된다. 이 구조에서도 마찬가지로 두개의 레지스터 그룹을 이용하여 한 그룹이 데이타 버스를 사용할 때 다른 한 그룹은 프로세서 엘리먼트에 데이타를 공급한다.

(그림 8)은 SAD를 계산하는 프로세스 엘리먼트의 내부 블럭도이다. 이 블럭은 하나의 참조블럭에 대한 SAD를 계산하는 블럭으로서 뺄셈기, 절대값을 계산하기 위한 XOR 블럭과 계산된 8비트 결과를 누적시키는 16비트 덧셈기로 구성된다.



(그림 7) SAD Unit 블럭도(1개의 참조블록 입력포트)
(Fig. 7) The block diagram of SAD unit with one input ports for candidate blocks

〈표 1〉 1단계에서 3개의 참조블럭 입력포트를 갖는 SAD에서 입력데이터의 흐름
〈Table 1〉 Data Flow of SAD unit with three input ports for candidate blocks at Step 1

Step 1	RAx Content			RBx Content			Operation	
Cycle	RA0	RA1	RA2	RB0	RB1	RB2	RAx	RBx
0	y(0,i)	y(4,i)	y(8,i)				L(0,1,2)	
1				y(0,i+1)	y(4,i+1)	y(8,i+1)	C	L(0,1,2)
2	y(4,i)	y(8,i)	y(12,i)				L(2) & S	C
3				y(4,i+1)	y(8,i+1)	y(12,i+1)	C	L(2) & S
4	y(1,i)	y(5,i)	y(9,i)				L(0,1,2)	C
5				y(1,i+1)	y(5,i+1)	y(9,i+1)	C	L(0,1,2)
6	y(5,i)	y(9,i)	y(13,i)				L(2) & S	C
7				y(5,i+1)	y(9,i+1)	y(13,i+1)	C	L(2) & S
8	y(2,i)	y(6,i)	y(10,i)				L(0,1,2)	C
9				y(2,i+1)	y(6,i+1)	y(10,i+1)	C	L(0,1,2)
10	y(6,i)	y(10,i)	y(14,i)				L(2) & S	C
11				y(6,i+1)	y(10,i+1)	y(14,i+1)	C	L(2) & S
12	y(3,i)	y(7,i)	y(11,i)				L(0,1,2)	C
13				y(3,i+1)	y(7,i+1)	y(11,i+1)	C	L(0,1,2)
14	y(7,i)	y(11,i)	y(15,i)				L(2) & S	C
15				y(7,i+1)	y(11,i+1)	y(15,i+1)	C	L(2) & S
16								C

(L(x) : Load RAx or Load RBx L(x) & S : Load RAx or Load RBx & Shift C : Caculate SAD)

〈표 2〉 2단계에서 3개의 참조블럭 입력포트를 갖는 SAD에서 입력데이터의 흐름
〈Table 2〉 Data Flow of SAD unit with three input ports for candidate blocks at step 2

Step 2	RAx Content			RBx Content			Operation	
Cycle	RA0	RA1	RA2	RB0	RB1	RB2	RAx	RBx
0	y(0,i)	y(2,i)	y(4,i)				L(0,1,2)	
1				y(0,i+1)	y(2,i+1)	y(4,i+1)	C	L(0,1,2)
2	y(2,i)	y(4,i)	y(6,i)				L(2) & S	C
3				y(2,i+1)	y(4,i+1)	y(6,i+1)	C	L(2) & S
4	y(4,i)	y(6,i)	y(8,i)				L(0,1,2)	C
5				y(4,i+1)	y(6,i+1)	y(8,i+1)	C	L(0,1,2)
6	y(6,i)	y(8,i)	y(10,i)				L(2) & S	C
7				y(6,i+1)	y(8,i+1)	y(10,i+1)	C	L(2) & S
8	y(1,i)	y(3,i)	y(5,i)				L(0,1,2)	C
9				y(1,i+1)	y(3,i+1)	y(5,i+1)	C	L(0,1,2)
10	y(3,i)	y(5,i)	y(7,i)				L(2) & S	C
11				y(3,i+1)	y(5,i+1)	y(7,i+1)	C	L(2) & S
12	y(5,i)	y(7,i)	y(9,i)				L(0,1,2)	C
13				y(5,i+1)	y(7,i+1)	y(9,i+1)	C	L(0,1,2)
14	y(7,i)	y(9,i)	y(11,i)				L(2) & S	C
15				y(7,i+1)	y(9,i+1)	y(11,i+1)	C	L(2) & S
16								C

〈표 3〉 3단계에서 3개의 참조블럭 입력포트를 갖는 SAD에서 입력데이터의 흐름
〈Table 3〉 Data Flow of SAD unit with three input ports for candidate blocks at step 3

Step 3	RAx Content			RBx Content			Operation	
Cycle	RA0	RA1	RA2	RB0	RB1	RB2	RAx	RBx
0	y(0,i)	y(1,i)	y(2,i)				L(0,1,2)	
1				y(0,i+1)	y(1,i+1)	y(2,i+1)	C	L(0,1,2)
2	y(1,i)	y(2,i)	y(3,i)				L(2) & S	C
3				y(1,i+1)	y(2,i+1)	y(3,i+1)	C	L(2) & S
4	y(2,i)	y(3,i)	y(4,i)				L(0,1,2)	C
5				y(2,i+1)	y(3,i+1)	y(4,i+1)	C	L(0,1,2)
6	y(3,i)	y(4,i)	y(5,i)				L(2) & S	C
7				y(3,i+1)	y(4,i+1)	y(5,i+1)	C	L(2) & S
8	y(4,i)	y(5,i)	y(6,i)				L(0,1,2)	C
9				y(4,i+1)	y(5,i+1)	y(6,i+1)	C	L(0,1,2)
10	y(5,i)	y(6,i)	y(7,i)				L(2) & S	C
11				y(5,i+1)	y(6,i+1)	y(7,i+1)	C	L(2) & S
12	y(6,i)	y(7,i)	y(8,i)				L(0,1,2)	C
13				y(6,i+1)	y(7,i+1)	y(8,i+1)	C	L(0,1,2)
14	y(7,i)	y(8,i)	y(9,i)				L(2) & S	C
15				y(7,i+1)	y(8,i+1)	y(9,i+1)	C	L(2) & S
16								C

〈표 4〉 1단계에서 1개의 참조블럭 입력포트를 갖는 SAD에서 입력데이터의 흐름
〈Table 4〉 Data Flow of SAD unit with one input port for candidate blocks at step 1

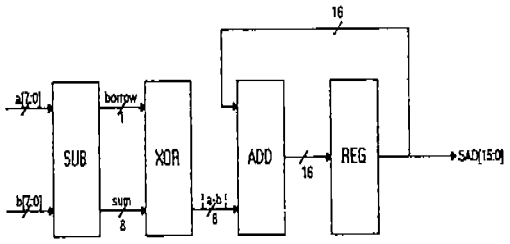
Step 1		RAx Content			RBx Content			Operation	
Cycle	Y[7:0]	RA0	RA1	RA2	RB0	RB1	RB2	RAx	RBx
0	y(0,i)	y(0,i)						L(0)	
1	y(0,i+1)				y(0,i+1)				L(0)
2	y(4,i)		y(4,i)					L(1)	
3	y(4,i+1)					y(4,i+1)			L(1)
4	y(8,i)			y(8,i)				L(2)	
5	y(8,i+1)						y(8,i+1)	C	L(2)
6	y(12,i)	y(4,i)	y(8,i)	y(12,i)				L(2) & S	C
7	y(12,i+1)				y(4,i+1)	y(8,i+1)	y(12,i+1)	C	L(2) & S
8	y(1,i)	y(1,i)						L(0)	C
9	y(1,i+1)				y(1,i+1)				L(0)
10	y(5,i)		y(5,i)					L(1)	
11	y(5,i+1)					y(5,i+1)			L(1)
12	y(9,i)			y(9,i)				L(2)	
13	y(9,i+1)						y(9,i+1)	C	L(2)
14	y(13,i)	y(5,i)	y(9,i)	y(13,i)				L(2) & S	C
15	y(13,i+1)				y(5,i+1)	y(9,i+1)	y(13,i+1)	C	L(2) & S
16	y(2,i)	y(2,i)						L(0)	C
17	y(2,i+1)				y(2,i+1)				L(0)
18	y(6,i)		y(6,i)					L(1)	
19	y(6,i+1)					y(6,i+1)			L(1)
20	y(10,i)			y(10,i)				L(2)	
21	y(10,i+1)						y(10,i+1)	C	L(2)
22	y(14,i)	y(6,i)	y(10,i)	y(14,i)				L(2) & S	C
23	y(14,i+1)				y(6,i+1)	y(10,i+1)	y(14,i+1)	C	L(2) & S
24	y(3,i)	y(3,i)						L(0)	C
25	y(3,i+1)				y(3,i+1)				L(0)
26	y(7,i)		y(7,i)					L(1)	
27	y(7,i+1)					y(7,i+1)			L(1)
28	y(11,i)			y(11,i)				L(2)	
29	y(11,i+1)						y(11,i+1)	C	L(2)
30	y(15,i)	y(7,i)	y(11,i)	y(15,i)				L(2) & S	C
31	y(15,i+1)				y(7,i+1)	y(11,i+1)	y(15,i+1)	C	L(2) & S
32									C

〈표 5〉 2단계에서 1개의 참조블럭 입력포트를 갖는 SAD에서 입력데이터의 흐름
〈Table 5〉 Data Flow of SAD unit with one input port for candidate blocks at step 2

Step 2		RAx Content			RBx Content			Operation	
Cycle	Y[7:0]	RA0	RA1	RA2	RB0	RB1	RB2	RAx	RBx
0	y(0,i)	y(0,i)						L(0)	
1	y(0,i+1)				y(0,i+1)				L(0)
2	y(2,i)		y(2,i)					L(1)	
3	y(2,i+1)					y(2,i+1)			L(1)
4	y(4,i)			y(4,i)				L(2)	
5	y(4,i+1)						y(4,i+1)	C	L(2)
6	y(6,i)	y(2,i)	y(4,i)	y(6,i)				L(2) & S	C
7	y(6,i+1)				y(2,i+1)	y(4,i+1)	y(6,i+1)	C	L(2) & S
8	y(8,i)	y(4,i)	y(6,i)	y(8,i)				L(2) & S	C
9	y(8,i+1)				y(4,i+1)	y(6,i+1)	y(8,i+1)	C	L(2) & S
10	y(10,i)	y(6,i)	y(8,i)	y(10,i)				L(2) & S	C
11	y(10,i+1)				y(6,i+1)	y(8,i+1)	y(10,i+1)	C	L(2) & S
12	y(1,i)	y(1,i)						L(0)	C
13	y(1,i+1)				y(1,i+1)				L(0)
14	y(3,i)		y(3,i)					L(1)	
15	y(3,i+1)					y(3,i+1)			L(1)
16	y(5,i)			y(5,i)				L(2)	
17	y(5,i+1)						y(5,i+1)	C	L(2)
18	y(7,i)	y(3,i)	y(5,i)	y(7,i)				L(2) & S	C
19	y(7,i+1)				y(3,i+1)	y(5,i+1)	y(7,i+1)	C	L(2) & S
20	y(9,i)	y(5,i)	y(7,i)	y(9,i)				L(2) & S	C
21	y(9,i+1)				y(5,i+1)	y(7,i+1)	y(9,i+1)	C	L(2) & S
22	y(11,i)	y(7,i)	y(9,i)	y(11,i)				L(2) & S	C
23	y(11,i+1)				y(7,i+1)	y(9,i+1)	y(11,i+1)	C	L(2) & S
24									C

〈표 6〉 3단계에서 1개의 참조블럭 입력포트를 갖는 SAD에서 입력데이터의 흐름
〈Table 6〉 Data Flow of SAD unit with one input port for candidate blocks at step 3

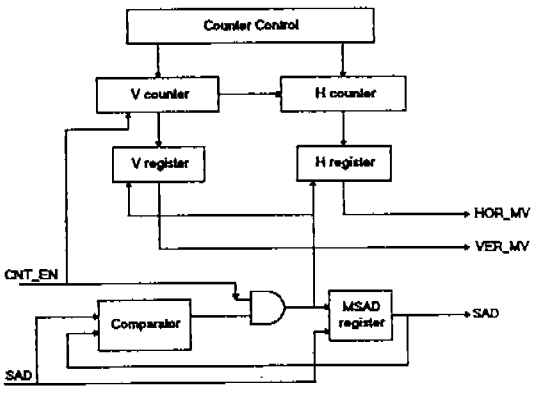
Step 3		RAx Content			RBx Content			Operation	
Cycle	Y[7:0]	RA0	RA1	RA2	RB0	RB1	RB2	RAx	RBx
0	y(0,i)	y(0,i)						L(0)	
1	y(1,i+1)				y(0,i+1)				L(0)
2	y(1,i)		y(1,i)					L(1)	
3	y(1,i+1)					y(1,i+1)			L(1)
4	y(2,i)			y(2,i)				L(2)	
5	y(2,i+1)						y(2,i+1)	C	L(2)
6	y(3,i)	y(1,i)	y(2,i)	y(3,i)				L(2) & S	C
7	y(3,i+1)				y(1,i+1)	y(2,i+1)	y(3,i+1)	C	L(2) & S
8	y(4,i)	y(2,i)	y(3,i)	y(4,i)				L(2) & S	C
9	y(4,i+1)				y(2,i+1)	y(3,i+1)	y(4,i+1)	C	L(2) & S
10	y(5,i)	y(3,i)	y(4,i)	y(5,i)				L(2) & S	C
11	y(5,i+1)				y(3,i+1)	y(4,i+1)	y(5,i+1)	C	L(2) & S
12	y(6,i)	y(4,i)	y(5,i)	y(6,i)				L(2) & S	C
13	y(6,i+1)				y(4,i+1)	y(5,i+1)	y(6,i+1)	C	L(2) & S
14	y(7,i)	y(5,i)	y(6,i)	y(7,i)				L(2) & S	C
15	y(7,i+1)				y(5,i+1)	y(6,i+1)	y(7,i+1)	C	L(2) & S
16	y(8,i)	y(6,i)	y(7,i)	y(8,i)				L(2) & S	C
17	y(8,i+1)				y(6,i+1)	y(7,i+1)	y(8,i+1)	C	L(2) & S
18	y(9,i)	y(7,i)	y(8,i)	y(9,i)				L(2) & S	C
19	y(9,i+1)				y(7,i+1)	y(8,i+1)	y(9,i+1)	C	L(2) & S
20									C



(그림 8) SAD 내부블럭도
(Fig. 8) The block diagram of SAD unit

(그림 9)는 움직임 벡터 생성기(Motion Vector Generator:MVG)의 블럭도이다. MVG는 9개의 블럭에 대한 SAD 값을 받고 이 중 가장 적은 SAD를 저장하고 있으면서 3단계가 끝나면 가장 작은 값을 갖는 SAD를 저장하여 움직임 벡터로 취한다. SAD unit에서 계산 되어온 결과가 MVG에 매 클럭마다 쉬프트되어 입력이 되고 카운터는 각 단계별로 다른 값이 증가한다. 즉, 1 단계는 4 씩 증가시키고, 2 단계에서는 2 씩, 3 단계에서는 1씩 증가하게 된다.

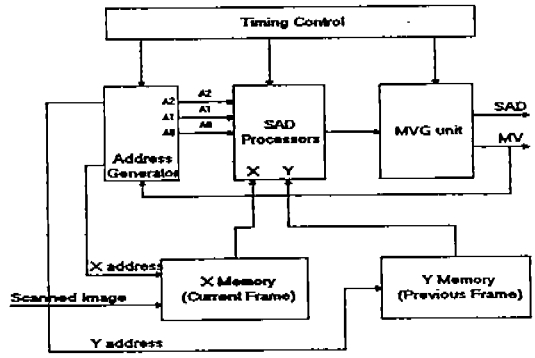
V counter는 링 카운터의 형태를 가지며 H counter에 연결이 되어 V counter 값이 최대치가 되면 H counter가 증가한다. 입력되는 SAD값이 순간적인 최소값(비교기의 출력)이 될 경우 V와 H Counter의 값이 V와 H 레지스터에 입력된다. 3 단계까지 연산이 끝난 후 V와 H 레지스터는 찾고자하는 움직임 벡터를 가진다. 각 스텝이 종료되면 H_Counter와 V_Counter는 1 단계에서는 2, 2 단계에서는 1씩 감소한다.



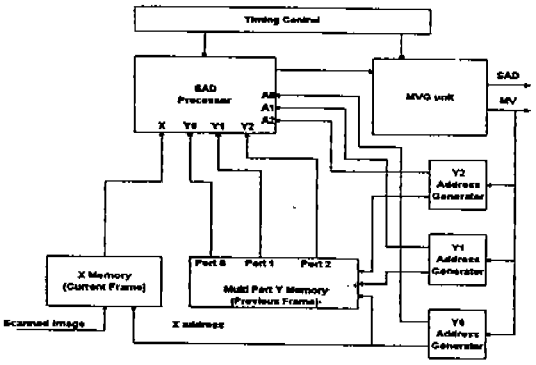
(그림 9) 움직임 벡터 생성기
(Fig. 9) The block diagram of motion vector generator

(그림 10)은 본 연구에서 제안한 한개의 참조블럭 입력 포트를 갖는 시스템의 전체 구성도이다. 지금까지 설명한 모든 동작은 주소생성기(Address Generator)와 Timing Control 부분에서 생성되는 제어신호에 의해서 동작한다. 움직임 벡터 생성기는 각 스텝이 끝나면 근사 예상 벡터를 주소생성기에 전달하여 다음 단계의 주소를 주소생성기가 생성할 수 있도록 한다. 이 구조에서 Timming Control 블럭이 생성한 제어신호들은 SAD프로세서 부분의 모든 프로세서 엘리먼트와 레지스터들을 동작시키는데 사용한다.

어드레스 생성은 매우 규칙적이므로 이를 위한 제어신호는 비교적 간단하게 구성될 수 있다. SAD에서 같은 열의 3개 프로세서 엘리먼트들은 제어신호를 공유한다. SAD의 내부에 레지스터들도 RA_x와 RB_x가 1 콜럼 차이를 두고 같은 제어신호에 의해 동작한다. 같은 열의 프로세스 엘리먼트들은 동일한 제어신호로 동작되므로, 전체 제어신호들은 매우 간단한 형태를 보인다. 참고문헌[3]의 일차원 시스톨릭 어레이같은 경우 내부에 각 프로세스 엘리먼트가 기준 블럭을 저장하는 레지스터 파일을 가지고 있어 상당히 칩의 크기가 커지는 요인이 있었지만 제안한 구조는 한 개의 레지스터 파일을 모든 프로세서가 공유하고 있다. 또한 각 레지스터 파일로 데이터를 공급해주는 레지스터는 단 6개로 할 수 있기 때문에 다른 구조에 비해서 매우 적은 로직으로 구현이 가능하여 칩의 크기 및 칩 가격면에서 상당히 유리한 요소를 지니고 있다. 입력 대역폭에 있어서도 기준블럭과 참조블럭을



(그림 10) 한 개의 참조블럭 입력포트를 갖는 시스템 구성도
(Fig. 10) The system architecture with one input port for candidate blocks



(그림 11) 세개의 참조블럭 입력포트를 갖는 시스템 구성도
(Fig. 11) The system architecture with three input ports for candidate blocks

입력할 수 있는 8비트 버스 두 개와 어드레스 버스만으로 구현이 가능하다.

(그림 11)은 본 연구에서 제안한 세개의 참조블럭 입력 포트를 갖는 시스템의 전체 구성도이다. 이 구조에서 (그림 10)과 다른 점은 참조블럭의 입력 포트가 세개이므로 멀티포트 메모리를 사용하여 3개 데이타가 동시에 SAD에 입력이 되어야 하므로 주소생성기가 3개가 필요하게 된다.

(그림 7)의 구조에서는 1 단계에서 i가 하나 증가할 때마다 32싸이클이 필요하고, 2단계에서는 24싸이클, 3단계에서는 20싸이클이 필요하다. 그러므로 각 단계별로 필요한 싸이클수를 계산하여 보면 $33\times8+25\times6+21\times5$, 즉 519싸이클이 필요하다. 또한 각 단계의 최종 출력을 위한 SAD의 쉬프트를 위해서 $9\times3+3=30$ 싸이클이 필요하다. 그러므로 한 기준 블럭의 움직임 추정을 하기 위해서는 549 싸이클이 필요하다. 이 구조에서 N×N 크기의 기준블럭을 사용할때 한 블럭의 움직임 벡터를 찾는데 걸리는 싸이클 수를 Z라하고 일반식으로 표현하면 다음과 같다.

$$Z=(N+\frac{N}{2}\times2)^2+\frac{(N+\frac{N}{2}\times2)}{2}+(N+\frac{N}{4}\times2)^2$$

$$+\frac{(N+\frac{N}{4}\times2)}{2}+(N+\frac{N}{8}\times2)^2+\frac{(N+\frac{N}{8}\times2)}{2}$$

$$+9\times3+3=\frac{125}{16}N^2+\frac{19}{8}N+30.$$

위의 식에서 첫번째부터 세번째 항은 각 단계의 탐색영역에서 데이타 입력에 필요한 싸이클과 데이타 열 변경에 필요한 싸이클수의 합이다. 다음 $9\times3+3$은 각 단계의 최종 출력에 요구되는 SAD의 쉬프트를 위한 클럭수이다. 예를 들어 블럭의 크기가 16×16일 때 $Z=(16+8\times2)^2+(16+8\times2)/2+(16+4\times2)^2+(16+4\times2)/2+(16+2\times2)^2+(16+2\times2)/2+9\times3+3$ 즉, $Z=2068$이 된다.

(그림 3)의 구조는 (그림 7)의 구조보다 간단하다. 기준블럭의 크기가 N×N일 때 i가 1씩 증가할 때 마다 2N싸이클이 필요하므로 일반식으로 표시하면 다음과 같다.

$$Z=2N\times[\frac{(N+\frac{N}{2}\times2)}{2}+\frac{(N+\frac{N}{4}\times2)}{2}$$

$$+\frac{(N+\frac{N}{8}\times2)}{2}]+9\times3+3=\frac{19}{4}N^2+30.$$

위의 식에서 첫 번째 항은 각 단계에서 탐색영역의 데이터를 입력하는데 요구되는 싸이클 수이며 $9\times3+3$은 각 단계의 최종 출력에 요구되는 SAD의 쉬프트를 위한 클럭수이다. 예를 들어 블록의 크기가 16×16일 때 $Z=2\times16\times[(16+16)/2+(16+8)/2+(16+4)/2]+9\times3+3$ 즉, $Z=1246$이 된다.

〈표 7〉은 참고문헌에서 언급된 구조들과 제안된 구조를 비교한 결과이며 제안한 구조가 다른 구조와 비교하여 입력에 요구되는 데이터 핀 수가 적으며 성능면에서도 다른 구조에 비해 우수함을 알 수 있다. 제안한 구조에서 움직임 벡터를 찾기위해 필요한 클럭수를 Z, 클럭 싸이클을 T_c라 하고, 프레임 크기를 X×Y, 기준 블럭의 크기를 N×N, 그리고 초당 R 프레임을 처리한다고 가정 할 때, 아래와 같은 수식이 성립하게 된다.

$$\frac{ZT_cXY}{N^2}\le\frac{1}{R}$$

따라서

$$T_c \le \frac{N^2}{ZXYR}$$

이 수식을 (그림 7)의 구조를 H.261 표준화면[5]인 CIF에 적용하여 보면(352×288 크기의 프레임, 16×16 블럭크기, 초당 30프레임) T_c는 40.7 ns보다는 적어야 한다. 즉, 24.6 MHz보다 높은 동작 주파수에서 움직여야 한다. 이는 현재의 CMOS 기술로 충분히 구현이 가능하며 다른 칩에 비하여 낮은 주파수에서도 같은 기능을 충분히 할 수 있다. (그림 3)의 구조를 가질 경우는 T_c는 67.6 ns보다 적은 값을 가져야 한다. 즉, 14.80MHz이상의 주파수에서 동작할 수 있다.

〈표 7〉 제안된 구조와 기존 구조의 성능 비교
〈Table 7〉 The comparision of performance between the architecture in this paper and proposed architecture

	완전검색 SIMD-Systolic[1]	2-D Systolic Array[2]	TSS 1-D Systolic Array[3]	TSS 고속 움직임 예측기	
				1 port	3 port
싸이클 수	1446	527	2924	2068	1246
데이타 핀의 수	24개	136개	18개	16개	32개

4. 결 론

본 연구에서는 삼단 검색 알고리즘을 기반으로 한 새로운 구조를 제안하였다. 기존의 삼단 검색 알고리즘은 규칙성을 찾기가 어려워 구현하기 힘들었고 앞 단계의 계산이 끝나고 후 다음 단계 계산이 가능하므로 병렬처리나 파이프라인이 어려웠다. 또한 큰 데이터 입력의 대역폭이 요구되었고 이를 해결하기 위해 I/O 핀 갯수의 증가를 동반하였다. 본 연구에서 데이터 입력의 대역폭을 최소화하는 동시에 데이터 입력 순서에 규칙성을 부여하는데 중점을 두고 구조를 제안하였다. 제안된 구조는 기존의 구조에 비하여 I/O 핀 갯수가 적을 뿐만 아니라 레지스터 갯수와 로직이 간단하며 성능면에서도 기존의 구조들을 크게 앞선다.

참 고 문 헌

[1] Chen-Mie Wu and Ding-Kuen Yeh, "A VLSI Motion Estimator for Video Image Compression", *IEEE trans. on Consumer Electronics*, Vol.39, No. 4, pp.837-845, November 1993.

[2] Eric Chan and Sethuraman Panchanathan "Motion Estimation Architecture for Video Compression", *IEEE trans on Consumer Electronics*, Vol.3, pp.292-297, August 1993.

[3] Yeu-Shen Jehng, Liang-Gee Chen and Tzi-Dar Chiueh, "A Motion Estimator for Low Bit-Rate Video Codec", *IEEE trans. on Consumer Electronics*, Vol.38, No.2, pp.60-69, May 1992.

[4] Alessandra Costa, Alessandro De Gloria, Paolo Faraboschi and Flilip Passaggio, "A VLSI Architecture for Hierachical Motion Estimation", *IEEE trans on Consumer Electronics*, Vol.41, pp.248-261, January 1995.

[5] CCITT Recommemtation H.261, "Video codec for audio-visual services at $p\times64$ Kbits/s", May 1989.

[6] Santanu Dutta, Wayne Wolf, "A Flexible Parallel Architecture Adapted to Block-Matcing Motion-Estimation Algorithms," *IEEE Trans. Circuits Syst. Video Technol.* vol.6, no.1, pp.74-86, Feb. 1996.

[7] Lai-Man Po, Wing-Chung Ma, "A Novel Four-Step Search Algorithm for Fast Block Motion Estimation," *IEEE Trans. Circuits Syst. Video Technol.* vol.6, no.3, pp.313-317, June 1996.

김 상 중

1949년 5월 15일생
1977년 2월 한양대학교 전자공학과 졸업
1980년 8월 연세대학교 전자공학과 졸업(석사)
1997년 2월 아주대학교 전자공학과 졸업(박사)
1977년 3월~1985년 4월 한국전자통신연구소 선임연구원
1985년 5월~현재 한국전자통신연구소 뉴미디어서비스연구실장, 책임연구원
주관심분야: 멀티미디어 통신, 영상 정보 처리, 컴퓨터 네트워크

김 용 길

1969년 7월 26일생
1995년 2월 아주대학교 전자공학과 졸업(학사)
1997년 2월 아주대학교 전기전자공학부 졸업(석사)
1997년~현재 대우 고등기술연구원 연구원
주관심분야: VLSI 설계, 멀티미디어 시스템

임 강 빈

1969년 4월 27일생
1992년 2월 아주대학교 전자공학과 졸업(학사)
1994년 2월 아주대학교 전자공학과 졸업(석사)
1994년 3월~현재 아주대학교 전기전자공학부 박사과정
주관심분야: 컴퓨터 구조, VLSI 설계, 영상 부호화, 멀티미디어 시스템 설계 등

김 용 득

1946년 1월 30일생
1971년 연세대학교 전자공학과 졸업
1973년 연세대학교 대학원(공학석사)
1978년 연세대학교 대학원(공학박사)
1973년~1974년 불란서 ESE 연구원
1978년~1980년 미국 Stanford대학교 연구교수 재직
1978년~현재 아주대학교 전기전자공학부 교수
주관심분야: 디지틀 시스템 하드웨어 응용, ISDN 응용 및 접속방안

정 기 현

1958년 10월 21일생
1990년 퍼듀대학교(미) 공학박사
1991년~1992년 현대반도체연구소 연구원
1992년~현재 아주대학교 전기전자공학부 교수
주관심분야: 컴퓨터 구조, VLSI 설계, 영상 서비스, 멀티미디어 시스템 등