論文96-33A-12-17

고속 실시간 처리 Full Search Block Matching 움직임 추정 프로세서

(A Real-time High Speed Full Search Block Matching Motion Estimation Processor)

柳 在 熙 *, 金 峻 虎 **

(Jaehee You and Joonho Kim)

요 약

Full Search Block Matching 알고리듬을 이용한 고속 움직임 추정 프로세서의 VLSI 아키텍처 및 구현방안이 제안되었다. 제안된 움직임 추정 프로세서는 동일한 PE들에 의한 병렬 연산과 함께 pipeline 방식을 사용하여 연산속도가 향상되었고, 다양한 응용분야에 따라 성능 및 하드웨어양의 조절이 쉽도록 설계되었다. 또한, embedded memory를 통해, image 데이터를 재 사용하여, 칩 핀 수를 감소시켰고, 동시에 PE 이용도를 100%로 만들어 throughput을 극대화하였다. 모든 PE는 규칙적이면서 동일한 데이터 처리구조를 갖기 때문에 VLSI 구현이 용이하며, 데이터 입출력시간을 PE 내부에서의 연산속도보다 느리게 함으로써 I/O bottleneck 문제를 해결하였다. 마지막으로, 로직 및 spice 시뮬레이션 결과와 함께 성능이 평가, 비교되었고, chip layout이 제시되었다.

Abstract

A novel high speed VLSI architecture and its VLSI realization methodologies for a motion estimation processor based on full search block matching algorithm are presented. The presented architecture is designed in order to be suitable for highly parallel and pipelined processing with identical PE's and adjustable in performance and hardware amount according to various application areas. Also, the throughput is maximized by enhancing PE utilization up to 100% and the chip pin count is reduced by reusing image data with embedded image memories. Also, the uniform and identical data processing structure of PE's eases VLSI implementation and the clock rate of external I/O data can be made slower compared to internal clock rate to resolve I/O bottleneck problem. The logic and spice simulation results of the proposed architecture are presented. The performances of the proposed architecture are evaluated and compared with other architectures. Finally, the chip layout is shown.

* 正會員, 弘益大學校 電子電氣工學部

(School of Electrical Engineering, Hongik University.)

** 正會員, 現代電子

(Hyundai Electronics, Industries Co. Ltd.)

※ 본 연구는 한국과학재단 출연과제 '고속 실시간 처리 full search block matching 움직임 추정 VLSI 프로세서 설계' (과제번호 : 941-0900-076-1)의 지원을 받아 수행되었음.)

接受日字:1995年3月15日, 수정완료일:1996年12月5日

I. 서 론

오늘날, 영상신호처리기술의 괄목할만한 성장에 힘입어 화상회의 시스템이나, 화상전화, 고선명 텔레비전 등의 상용화를 위한 하드웨어 개발이 경쟁적으로 진행되고 있다. 이러한 영상신호처리 시스템의 개발에서 가장 어려운 점 중의 하나는 방대한 영상데이터의 실시간 압축처리 문제이다. 영상데이타 압축을 위한 방법에 있어서는, 한 영상 프레임 내에서의 공간적 redundancy를 제거하는 DCT (Discrete Cosine Transform)와 프레임과 프레임간의 시간적 redundancy를

제거하는 움직임 추정 기법 (Motion Estimation (ME))이 사용되고 있다.

위 방법중 움직임 추정기는 화상 부호화 시스템 중에서 가장 복잡하고 많은 연산 량을 필요로 하기 때문에 이의 효율적인 하드웨어의 개발이 매우 중요시되어 왔다. 움직임 추정기 개발 초기에는 programmable general signal processor [1-3]을 바탕으로 연구가 진행되었으나, 느린 동작속도로 인해 실시간 처리에 어려움이 많았다. 이후 최근에는 실시간 고속처리를 위해 알고리듬의 병렬성을 최대로 이용 가능한 어레이 프로세서 [4-6]들이 연구되고 있다.

이 분야에서 현재까지 발표된 움직임 추정기들을 살펴보면, 2차원 systolic array 및 shift register를 바탕으로 하는 [4]는 PE(Processing element)간의 데이터 교환이 세 방향으로 이루어져야 하므로 이로 인한 interconnection과 제어가 너무 복잡하고 data-flow방식을 이용한 [5]는 병렬 처리 시스템의 throughput을 향상시키는데 한계가 있다. 또, 1차원 systolic array방식을 사용한 [6]은 하나의 영상데이터 블록을 처리하는데 걸리는 시간이 상대적으로 길고, PE 이용도가 낮으며, 많은 shift register가 요구되는 단점이 있다. 특히, [4-6] 모두 PE의 수가 영상 데이터 블록의 크기 또는 탐색범위에 의해 고정되기 때문에 성능 및 하드웨어양의 조절이 어려워 MPEG 등의 다양한 영상처리 응용분야에 적용하기 어려운 점이 있다.

본 논문에서는 다른 알고리듬에 비해 정확도 면에서 가장 좋은 성능을 제공하고, 데이터 구조가 규칙적인 Full Search Block Matching Algorithm (FBMA)을 바탕으로, 1) 규칙적이고 동일한 하드웨어 구조에 의한 VLSI 구현용이, 2) 움직임벡터 계산간의 휴지시간 제거를 통한 100% PE utilization 및 throughput 향상, 3) 규칙적이고 느린 I/O rate, 4) 적은 양의 I/O 핀 수, 5) 성능과 하드웨어 양의 조절용이 등을 이룰 수 있는 움직임 추정기의 VLSI 아키텍처 및 구현방안이 제안되었다.

본 논문의 구성은 II장에서 FBMA를 소개하고, III장에서는 본 논문에서 제안된 VLSI 아키텍처와 구현방안이 설명되었다. IV장에서는 제안된 아키텍처의 성능 평가 및 다른 아키텍처와의 비교, 장점 등이 소개되고, V장에서는 중요회로의 로직 및 SPICE 시뮬레이션 결과와 칩의 전체 layout을 나타내었다.

II. Full Search Block Matching Algorithm.

FBMA는 간략하게 식 (1)으로 표현될 수 있다.

$$EV(h,\ v) = \sum_{t=0}^{N-1}\sum_{s=0}^{N-1} |\ PF(s+h,\ t+v) - CF(s,\ t)\ | \qquad (-p \le h,\ v < p)$$

where, PF : previous frame, CF : current frame.

Block size : $N \times N$, Search range : p.

$EV(h, v)$: absolute error between $PF(s+h, t+v)$ and $CF(s, t)$

MEDV : (h, v) when $EV(h, v)$ is minimum.

식 (1)에서 EV(h, v)는 CF 블록의 각 화소 데이터와 이 CF 블록으로부터 수평으로 h, 수직으로 v 화소만큼 떨어져 있는 PF 블록의 각 화소 데이터간 차이의 절대값을 모두 더한 값으로, 이들 중 가장 작은 EV(h, v)의 (h, v)를 벡터로 표시한 것이 MEDV (Minimum Error Displacement Vector) 이다. MEDV는 위 CF 블록이 어떤 PF 블록으로부터 이동했는가를 나타내는 정보, 즉 움직임벡터가 된다.

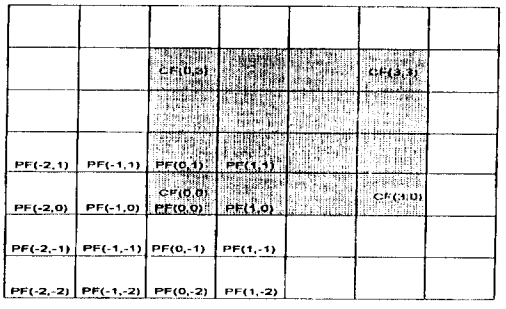


그림 1. N = 4, p = 2 일 때의 CF 블록과 PF 블록
Fig. 1. CF block and PF blocks when N = 4, p = 2.

식 (1)에서처럼 탐색범위가 p 일 때 각 CF 블록당 비교되는 PF 블록의 수는 2p x 2p = $4p^2$가 된다. 그림 1에 N = 4, p = 2 일 때 한 CF 블록과 이와 비교되는 PF 블록들을 나타내었다. 여기서 각 PF 블록의 위치는 그 블록의 가장 왼쪽, 위 화소로 표현하였다.

III. VLSI 아키텍처

그림 2에 본 논문에서 제안된 움직임 추정기의 전체 블록도를 나타내었다. 움직임 추정연산에 필요한 많은 양의 PF 및 CF data는 효율적으로 PE에 전달되어야

하는데, 하나의 CF 블록에 대한 MEDV 연산(이하 macro block 연산) 초기에 일시에 시스템 외부에서 받아들일 경우, 많은 양의 interconnection과 chip pin을 필요로 하게되며, 이를 감소시킬 경우, macro block 연산간에 많은 지연시간이 발생하고, 하드웨어의 이용도가 떨어져, throughput을 감소시키게 된다. 따라서 제안된 아키텍처에서는 적은 수의 pin을 사용하여, serial 방식으로 PF data를 이전 macro block연산과 overlap시켜, internal PF buffer 및 external PF buffer에 입력시키고, macro block 연산 종료 직후 일시에 shift resister array에 입력시켜, 다음 macro block연산을 시작한다. Internal PF buffer는 다음 CF 블록의 움직임벡터 초기연산에 필요한 PF 데이터를 입력 핀을 통해 받아들이고, external PF buffer memory는 현재의 CF 블록에 대한 움직임벡터 계산에 필요한 PF 데이터를 계속 PE 어레이로 공급하면서 동시에 다음 CF 블록의 움직임벡터 계산에 필요한 PF 데이터 중에서 internal PF buffer에 이미 저장된 PF 데이터를 제외한 나머지 PF 데이터들을 받아들인다. 연산되는 PF data는 shift register array에 의해 shift되면서 각 PE에 올바른 데이터를 공급하여, EV(h, v) 연산을 수행하게 된다.

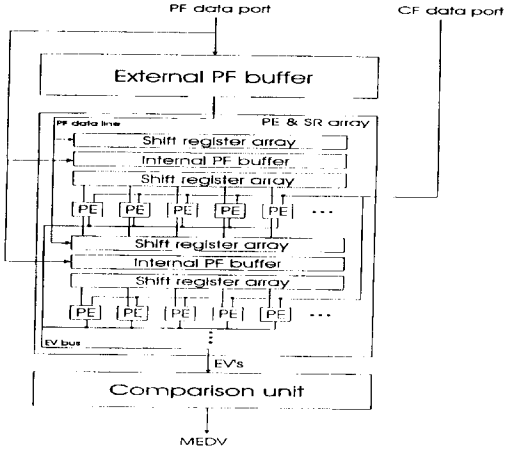


그림 2. 제안된 움직임 추정기의 전체 블록도
Fig. 2. Overall block diagram of proposed motion estimation processor.

한 번 입력된 PF data는 tracking range 내의 PF macro block이 서로 중복된 데이터로 구성되어 있어, 효율적으로 재 사용될 필요가 있다. 이를 위해 한 번 입력된 데이터는 shift resister array 간에 전달되며, 서로 다른 PF 블록을 구성하며 연산을 수행하게 된다. 이때 CF 데이터는 모든 PE에서 동일한 것이 사용되기 때문에 각 PE로 broadcasting 한다. 이에 대한 자세한 사항은 III. 1. 1에서 소개된다.

제안된 아키텍처에 있어 모든 PE는 동일한 구조로 이루어져 있어, 용이하게 VLSI 구현이 가능하고, 각 PE 내부의 수행연산 내용도 동일하여 간단한 제어로 구현 가능하다. 계산이 종료된 EV(h, v)들은 각 PE에 저장되었다가 serial 방식으로 EV bus를 통해 comparison unit으로 보내어진다. Bus를 통해 EV가 전달됨으로써, PE와 comparison unit과의 interconnection 복잡도를 크게 감소시킬 수 있다. 이후 EV(h, v) 비교를 통해 MEDV 즉 움직임벡터를 찾아낸다.

1. PE 어레이

1) Data flow

EV(h, v)를 구하기 위한 연산은 CF 데이터와 PF 데이터간 차이를 계산하고 이의 절대값들을 accumulation하는 과정으로 이루어지며, 이는 PE에서 수행된다. 식 (1)에 나타낸 움직임 추정 알고리듬을 실시간 처리하기 위해서는 여러 개의 PE로 동시에 여러 개의 EV(h, v)들을 계산하는 병렬처리가 필요하다. 제안된 아키텍처에서는 데이터의 이동 구조상 유리한 (h, v)에 의한 병렬처리를 사용하였다.

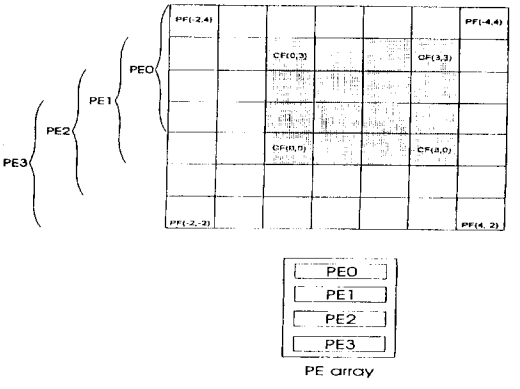


그림 3. PE array 구성
Fig. 3. PE array structure.

모든 EV(h, v)에 대해서 완전병렬처리를 하면, PF 블록이 모두 $4p^2$ 개이므로 PE 또한 $4p^2$ 개가 필요하다. 그런데, 완전병렬처리는 가장 빠른 연산속도를 얻을 수는 있지만, 요구되는 PE 수가 많으므로 칩의 크기가 구현 불가능할 수 있다. 따라서, 각 응용분야에

따라 요구되는 연산속도와 칩 면적을 고려하여 적절한 PE 수로 직, 병렬처리를 하는 것이 바람직하다. 각 PE가 계산하게 되는 EV(h, v)의 수를 n이라고 하면, n은 '총 PF 블록수/총 PE 수' 가 된다. 제안된 아키텍처에서는 응용분야에 따라, n이 조절 가능하게 하여, 하드웨어 양과 연산속도를 자유롭게 조절 가능하게 함으로써, 예를 들어 MPEG 등의 여러 영상처리 규격을 전부 동일한 아키텍처로 구현 가능하도록 하였다. 본 논문에서 제안한 아키텍처와 연산구조는 그림 1에서와 같이 인접한 PF 블록들이 서로 PF 데이터를 공유하는 점을 최대한으로 이용하였으며, 이에 따른 PE 배치방법을 그림 3에 그림 1의 경우를 예로 들어 나타내었다.

그림 3에 나타냈듯이 수직방향의 PE 개수는 2p개이며, 각 PE에서 n개의 EV(h, v)가 interleaving 방식으로 처리되는 과정을 표 1에 나타내었다. 표 1에서 PF 데이터의 처리순서는 PF 데이터의 단 한번 입력으로 n개의 PF 블록에 대한 EV(h, v) 계산이 가능함을 보여준다. 또, 각 PE에서 n개의 PF 블록에 대한 EV(h, v)를 interleaving 방식으로 계산하므로 PE간 PF 데이터 전송시간이 n 클럭동안의 여유가 있다. CF 데이터는 표 1에서 나타낸 바와 같이, 모든 PE가 동일한 데이터를 사용하므로 각 PE로 broadcasting되며, 한 번 입력되면, 모든 PE에서 같은 데이터가 n 클럭 동안 사용된다. 이러한 느린 데이터 broadcasting 간

표 1. 각 PE에서의 연산과정
Table 1. Computation sequences of each PE.

clock	PE 0	PE 1	PE 2	PE 3
0	| PF(-2, 4) - CF(0, 3) |	| PF(-2, 3) - CF(0, 3) |	| PF(-2, 2) - CF(0, 3) |	| PF(-2, 1) - CF(0, 3) |
1	| PF(-1, 4) - CF(0, 3) |	| PF(-1, 3) - CF(0, 3) |	| PF(-1, 2) - CF(0, 3) |	| PF(-1, 1) - CF(0, 3) |
2	| PF(0, 4) - CF(0, 3) |	| PF(0, 3) - CF(0, 3) |	| PF(0, 2) - CF(0, 3) |	| PF(0, 1) - CF(0, 3) |
3	| PF(1, 4) - CF(0, 3) |	| PF(1, 3) - CF(0, 3) |	| PF(1, 2) - CF(0, 3) |	| PF(1, 1) - CF(0, 3) |
4	| PF(-1, 4) - CF(1, 3) |	| PF(-1, 3) - CF(1, 3) |	| PF(-1, 2) - CF(1, 3) |	| PF(-1, 1) - CF(1, 3) |
5	| PF(0, 4) - CF(1, 3) |	| PF(0, 3) - CF(1, 3) |	| PF(0, 2) - CF(1, 3) |	| PF(0, 1) - CF(1, 3) |
6	| PF(1, 4) - CF(1, 3) |	| PF(1, 3) - CF(1, 3) |	| PF(1, 2) - CF(1, 3) |	| PF(1, 1) - CF(1, 3) |
7	| PF(2, 4) - CF(1, 3) |	| PF(2, 3) - CF(1, 3) |	| PF(2, 2) - CF(1, 3) |	| PF(2, 1) - CF(1, 3) |
8	| PF(0, 4) - CF(2, 3) |	| PF(0, 3) - CF(2, 3) |	| PF(0, 2) - CF(2, 3) |	| PF(0, 1) - CF(2, 3) |
9	| PF(1, 4) - CF(2, 3) |	| PF(1, 3) - CF(2, 3) |	| PF(1, 2) - CF(2, 3) |	| PF(1, 1) - CF(2, 3) |
10	| PF(2, 4) - CF(2, 3) |	| PF(2, 3) - CF(2, 3) |	| PF(2, 2) - CF(2, 3) |	| PF(2, 1) - CF(2, 3) |
11	| PF(3, 4) - CF(2, 3) |	| PF(3, 3) - CF(2, 3) |	| PF(3, 2) - CF(2, 3) |	| PF(3, 1) - CF(2, 3) |
12	| PF(1, 4) - CF(3, 3) |	| PF(1, 3) - CF(3, 3) |	| PF(1, 2) - CF(3, 3) |	| PF(1, 1) - CF(3, 3) |
13	| PF(2, 4) - CF(3, 3) |	| PF(2, 3) - CF(3, 3) |	| PF(2, 2) - CF(3, 3) |	| PF(2, 1) - CF(3, 3) |
14	| PF(3, 4) - CF(3, 3) |	| PF(3, 3) - CF(3, 3) |	| PF(3, 2) - CF(3, 3) |	| PF(3, 1) - CF(3, 3) |
15	| PF(4, 4) - CF(3, 3) |	| PF(4, 3) - CF(3, 3) |	| PF(4, 2) - CF(3, 3) |	| PF(4, 1) - CF(3, 3) |
16	| PF(-2, 3) - CF(0, 2) |	| PF(-2, 2) - CF(0, 2) |	| PF(-2, 1) - CF(0, 2) |	| PF(-2, 0) - CF(0, 2) |
17	| PF(-1, 3) - CF(0, 2) |	| PF(-1, 2) - CF(0, 2) |	| PF(-1, 1) - CF(0, 2) |	| PF(-1, 0) - CF(0, 2) |
18	| PF(0, 3) - CF(0, 2) |	| PF(0, 2) - CF(0, 2) |	| PF(0, 1) - CF(0, 2) |	| PF(0, 0) - CF(0, 2) |
19	| PF(1, 3) - CF(0, 2) |	| PF(1, 2) - CF(0, 2) |	| PF(1, 1) - CF(0, 2) |	| PF(1, 0) - CF(0, 2) |
. . .	. . .	. . .	. . .	. . .
60	| PF(-2, 1) - CF(3, 0) |	| PF(-2, 0) - CF(3, 0) |	| PF(-2,-1) - CF(3, 0) |	| PF(-2,-2) - CF(3, 0) |
61	| PF(-1, 1) - CF(3, 0) |	| PF(-1, 0) - CF(3, 0) |	| PF(-1,-1) - CF(3, 0) |	| PF(-1,-2) - CF(3, 0) |
62	| PF(0, 1) - CF(3, 0) |	| PF(0, 0) - CF(3, 0) |	| PF(0,-1) - CF(3, 0) |	| PF(0,-2) - CF(3, 0) |
63	| PF(1, 1) - CF(3, 0) |	| PF(1, 0) - CF(3, 0) |	| PF(1,-1) - CF(3, 0) |	| PF(1,-2) - CF(3, 0) |

격은 VLSI 구현시 고속으로 global interconnection line을 동작시킬 필요가 없어, 구현이 용이하다.

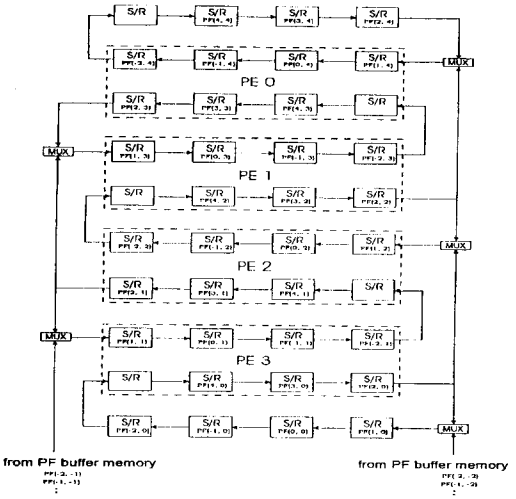


그림 4. PF data flow
Fig. 4. PF data flow.

그림 4에 표 1에 나타낸 연산순서에 따른 인접된 PE에 위치하는 shift register array내의 PF data 이동 구조를 나타내었다. 그림 4의 mux 들은 (2pN / 수평으로 어레이된 PE 수) 클럭마다 위쪽과 아래쪽 패스를 번갈아 가며 연결시켜 준다. 각 PE는 그림 4의 shift register 안에 나타낸 PF 데이터들을 연산의 초기 값으로 하여 그림 4에서 화살표로 나타낸 방향으로 PE간에 PF 데이터들을 이동시키며 EV(h, v)를 계산한다.

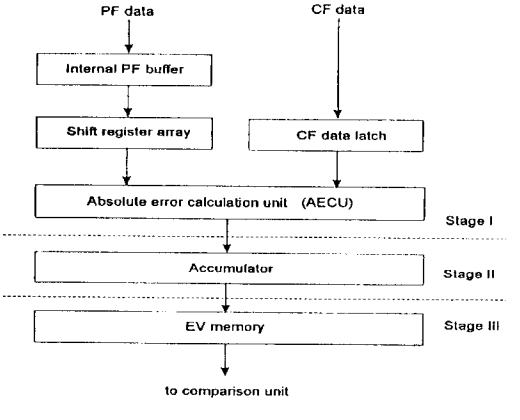


그림 5. PE의 블록도
Fig. 5. PE block diagram.

2) PE구조

그림 5에 PE의 블록 다이어그램을 나타내었다. 그림 5처럼 PE는 throughput 향상을 위해 세 단계로 pipeline되어 있다. 본 절에서는 각 블록별 VLSI 구현 방안을 소개한다.

(1) CF data latch와 internal PF buffer

CF data latch와 internal PF buffer는 CF와 PF 데이터 각각에 있어, 연산될 데이터를 latch 시킨다. CF 데이터는 global interconnection에 의해 전송되어 긴 전송시간을 가지므로 다음 계산에 쓰일 CF 데이터를 미리 받아들이도록 하였다. 이 CF data latch는 n 클럭마다 저장된 내용을 출력하도록 제어된다. PF buffer는 이전 macro block연산이 진행 중에 외부 칩 pin으로 부터 serial 방식으로 PF data를 저장하게 된다. 기존의 SRAM 구조로 구현가능하며, 이전 macro block 연산이 종료한 직후 shift register array로 데이터가 일시에 출력되며, 이후 위와 같은 동작을 반복하게 된다. PF 데이터는 칩 pin으로부터 serial방식으로 데이터를 입력받으므로, 적은 양의 interconnection으로 전송 가능하다.

(2) Shift register array

그림 4처럼 PE간에는 PF 데이터가 전송되어야 하며, 이를 위한 shift register를 그림 6에 나타내었다. Shift register는 각 움직임벡터 계산의 첫 번째 클럭에 PF buffer로부터 PF 데이터를 받은 뒤 그림 4에서 화살표로 나타낸 방향에 따라 이들을 이동시킨다. 그림 6에서 나타낸바와 같이 pass-gate 1을 통해 internal PF memory로부터 PF 데이터를 받아들이며, pass-gate 2는 다른 array내의 shift register로부터 PF 데이터를 입력받게 된다.

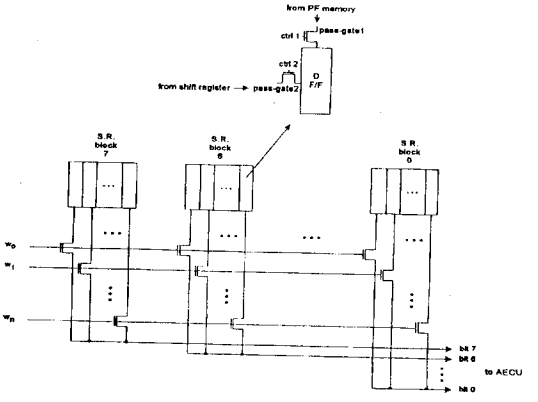


그림 6. Shift register 구조도
Fig. 6. Shift register structure.

(3) AECU

그림 7에 나타낸 AECU (Absolute Error Calculation Unit)는 PF 데이터와 CF 데이터간 차이의 절대값을 계산하는 부분이다. 두 데이터 차이의 절대값은 2의 보수를 바탕으로 adder 두개와 mux를 사용하여 구현할 수 있다. 그림 7에서 adder 1은 두 수의 차이 (A - B)를 구하고 adder 2는 두 수의 차이가 음수인 경우에 절대값을 얻는데 사용되며, 본 논문에서는 pipeline rate를 올리기 위해 CLA(Carry Look-ahead Adder)를 이용하였다. 그림 7의 mux는 adder 1의 출력에서 부호 bit이 양수 (0)이면 그 출력값을 그대로 보내고, 부호 bit이 음수 (1)이면 adder 2를 거친 출력값을 보낸다.

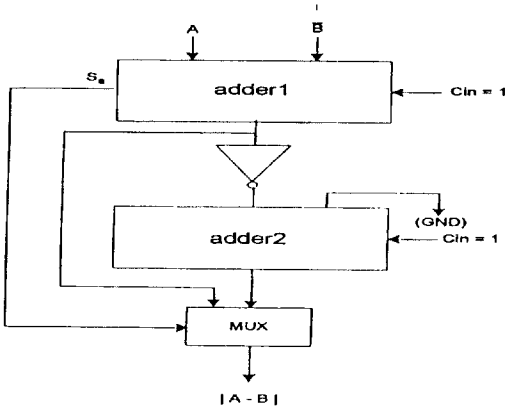


그림 7. 절대값 계산 하드웨어구조도
Fig. 7. Absolute error calculation unit.

Adder 1의 출력이 음수일 경우는 절대값을 얻기 위하여 2의 보수를 취해야 하는데 이때는 총 두개의 adder를 직렬로 거쳐야 하므로 PE의 연산속도 저하를 가져올 수 있다. 그러나, 2의 보수를 구하는 과정이 단지 adder 1 출력을 inversion 한 값에 +1만 더하면 되므로, 하드웨어를 매우 간소화시킬 수 있다. 먼저 inversion은 인버터로 간단히 구현된다. 그리고 adder 2는 그림 7처럼 첫 번째 carry, Cin에 '1'을 입력시키면 adder 2의 한쪽 입력의 모든 bit은 항상 '0' 이므로 일반적인 CLA 계산식 [7]에서 B_i가 '0'이 된다. 이를 이용하여 간략화된 adder 2의 논리식을 식 (2)에 나타내었다.

$$G_i \text{ (generate term)} = A_i B_i = 0$$

$$P_i \text{ (propagate term)} = A_i \oplus B_i = A_i$$

$$S_i = C_{i-1} \oplus P_i = C_{i-1} \oplus A_i$$

$$C_1 = G_1 + P_1C_0 = A_1$$

$$C_2 = G_2 + P_2G_1 + P_2P_1C_0 = A_2A_1$$

$$\vdots$$

$$C_i = A_iA_{i-1}A_{i-2} \cdots A_1 \quad (2)$$

식 (2)에 나타난 바와 같이, 단지 +1을 위해서는 보통의 CLA에서 요구되는 G_i, P_i를 위한 회로가 필요없으며 C_i를 계산하기 위한 회로도 간소해진다. 따라서 그림 7의 AECU는 두개의 adder가 직렬로 연결되어 있으나, adder 2가 간소화되기 때문에 높은 pipeline rate가 구현 가능하다.

(4) Accumulator

EV(h, v)는 모든 CF 데이터와 PF 데이터간 차이의 절대값들을 더한 값이므로 accumulation이 필요하다. 이를 위한 adder는 PE가 n개의 PF 블록에 대한 EV(h, v)를 interleaving 방식에 의해 순차적으로 계산하므로 한 개만 있으면 충분하다.

(5) EV memory

움직임벡터를 구하려면 각 PE에서 계산된 EV(h, v)들이 서로 비교되어야 한다. 본 논문에서는 PE 어레이 밖의 comparison unit를 통하여 위 연산을 수행한다. 만일 각 PE에서 계산된 EV(h, v)를 global comparison unit으로 전송할 때에 PE마다 각각 독립적인 데이터라인을 사용한다면 하드웨어 면적을 크게 증가시킨다. 따라서 모든 PE가 bus를 통하여 순차적으로 EV(h, v) 데이터를 전송한다. PE 내부의 EV memory는 계산된 EV(h, v)를 전송할 때까지 EV(h, v) 데이터를 저장한다.

(6) Comparison unit

움직임벡터를 구하기 위해서는 계산된 EV(h, v)들을 비교하여 최소값을 구할 필요가 있다. 이를 위하여 PE마다 comparator를 두는 방법은 각 comparator의 동작 시간이 PE 내의 EV(h, v) 계산시간에 비해 매우 짧아 하드웨어 효용도를 떨어뜨리고 칩면적을 증가시키게 된다. 따라서 그림 2처럼 PE 어레이 외부에서 한 개의 global comparison unit을 비교기를 사용하여 순차적으로 비교연산을 수행하였다. 각 PE에서 comparison unit으로 EV(h, v)를 전송할 때 PF 블록의 CF 블록에 대한 변위정보, 즉 (h, v) 벡터값도 MEDV 탐색을 위해 같이 전송할 필요가 있으므로 또 다른 데이터라인이 추가로 소요된다. 따라서, 이를 해

결하기 위해 각 PF 블록의 위치에 관한 정보를 보내는 대신에 미리 정해진 PF 블록의 순서에 의해 EV(h, v)를 전송하고 그 순서대로 comparison unit에서 (h, v) 벡터를 발생시키도록 하여 칩 면적을 감소시켰다. 그림 8에 comparison unit을 나타내었다. 그림 8의 displacement vector generator는 정해진 PF 블록 순서대로 (h, v) 벡터를 발생시키는데, $2(1 + \log_2 p)$ bit 카운터를 사용하였다. MEDV를 구하는 방법은 다음과 같다. 차례대로 전송되어 오는 EV(h, v)가 latch 1에 저장되어 있는 그때까지의 최소 EV(h, v)와 비교되고, 만약 새로운 EV(h, v)가 더 작으면 스위치 1, 2가 닫히어 새로운 EV(h, v)와 그때의 (h, v) 값이 latch 1, 2에 각각 새로 저장된다. 모든 비교 연산 ($4p^2$ 번)이 끝난 후에 latch 1의 값이 바로 최소 EV(h, v)가 되고, latch 2의 값이 최종적으로 구하고자 하는 움직임 벡터 (h, v)가 된다.

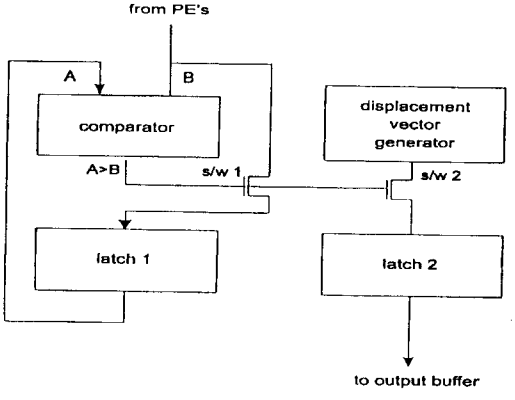


그림 8. Global comparison unit
Fig. 8. Global comparison unit.

2. 시스템 timing diagram

그림 9에 위에서 설명한 각 블록들의 동작에 대한 전체적인 타이밍도를 나타내었다. 그림 9에서 구간 ①은 PF 데이터를 칩 외부로부터 받아들이는 기간을, 구간 ②는 각 PE에서 EV(h, v)를 계산하는 시간을, 구간 ③은 comparison unit에서 각 PF 블록의 EV(h, v)들을 비교하여 MEDV를 찾는 시간을 나타낸다.

Ⅳ. 성능평가

1. 성능평가

본 논문에서 제안한 움직임 추정기는 연산구조가 파이프라인 방식으로 되어 있으므로 throughput은 그중 가장 긴 처리시간이 걸리는 stage에 의해 결정되는데 설계결과 이 stage는 PE 내부의 CF 데이터와 PF 데이터간 차이의 절대값을 구하는 부분이다. 따라서 이 연산에 걸리는 시간을 단위 클럭으로 하고 한 개의 움직임벡터를 계산하는데 필요한 시간인, 출력 사이클 타임, OCT를 구하면 식 (3)과 같다.

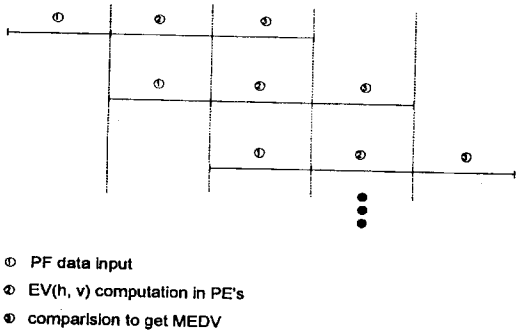


그림 9. 전체 동작 타이밍
Fig. 9. Overall timing diagram.

$$OCT = \text{한 PF 블록의 화소수} \times \text{PE당 처리할 PF 블록 수} = N^2 \times n = nN^2 \text{ clocks} \quad (3)$$

이제 하나의 프레임 내에 존재하는 CF 블록의 총수를 m이라고 하면, 이 프레임 전체에서 움직임벡터를 모두 계산하는데 걸리는 시간은 $m \times OCT = mn(N^2)$ clocks 이다.

표 2. 다른 아키텍처와의 성능비교
Table 2. Performance comparisons with other architectures.

아키텍처	PE 수	adder 수	comparator 수	데이터 핀수	tIDLE	움직임벡터 수/sec
[2]	256	528	1	136	16	1/512 x f
[4]	256	527	1	24	171	1/961 x f
[5]	256	511	1	84	496	1/992 x f
본 논문	256	512	1	24	0	1/256 x f
	64	128	1	24	0	1/1024 x f

tIDLE : 움직임 벡터 연산간의 휴지시간
f : Clock Frequency

표 2에 N = 16, p = 8 일 때 본 논문의 아키텍처와 다른 아키텍처의 성능비교 결과를 보였다. 표 2를 보면 본 논문의 아키텍처는 표 2의 (1), (2), (3) 과 비슷한

PE 및 adder 수를 가질 때 핀 수는 (2), (3)와 (1)의 중간정도이나, (1), (2), (3)이 모두 가지는 움직임벡터 계산간의 휴지시간이 전혀 없고, 단위시간당 구할 수 있는 움직임벡터의 수(throughput)가 훨씬 많음을 알 수 있다.

2. 제안된 아키텍처의 특징 및 장점

다른 아키텍처들과의 비교를 통한 본 논문의 아키텍처의 특징 및 장점은 다음과 같다.

본 논문의 아키텍처는

(1) 움직임 추정 연산과 I/O 연산이 중첩됨으로 인해 움직임벡터 계산간의 휴지시간이 제거되어 PE의 이용도가 100%이므로 throughput을 극대화시킬 수 있다. 이는 특히 HDTV의 경우와 같이 한 프레임의 블록수가 많을수록 효율적인 연산이 수행 가능하다.

(2) 요구되는 성능에 따라 PE 수를 용이하게 조절할 수 있어, 계산속도와 하드웨어 양의 조절이 용이하므로, 여러 응용분야에 따라 가장 효율적인 구현이 가능하다. 예를 들어 [5]의 수평방향으로 PE 개수가 N이어야 하는 제약조건 없이, 2차원 어레이 구조가 가능하여 FBMA의 병렬성을 최대한으로 이용할 수 있다.

(3) 규칙적이고 느린 I/O 및 이로 인한 적은 요구칩 핀 수 때문에 VLSI 구현이 용이할 뿐 아니라, 외부 image 메모리 구조 및 제어를 단순화시킬 수 있다.

(4) 동일한 PE 및 연산구조로 인해, VLSI 구현 및 제어가 용이하고, 각 하드웨어 구성부문 사이의 순차적인 데이터 이동으로 interconnection의 양이 적다.

(5) 약간의 수정으로 fractional-pel 단위의 움직임 추정 프로세서 [8]에도 수월하게 적용할 수 있다.

V. VLSI 구현

III장에서 설명한 아키텍처를 바탕으로 PF 및 CF 데이터 길이가 8 bit, CF 블록의 크기(N x N)가 16 x 16, 탐색범위(p)가 8인 움직임 추정기를 설계하였다. 설계된 움직임 추정기는 4 x 16 PE로 구성되었다.

설계과정은 먼저 설계된 아키텍처에 따라 top-down 방식에 의해 모듈을 설계한 후, MAGIC을 이용해 layout을 하고 routing을 수행하였다. 포스트 시뮬레이션을 위해 layout 데이터로부터 회로를 다시 추출한 후, IRSIM과 SPICE를 이용하여 로직 및 타이밍 시뮬레이션을 수행하였다.

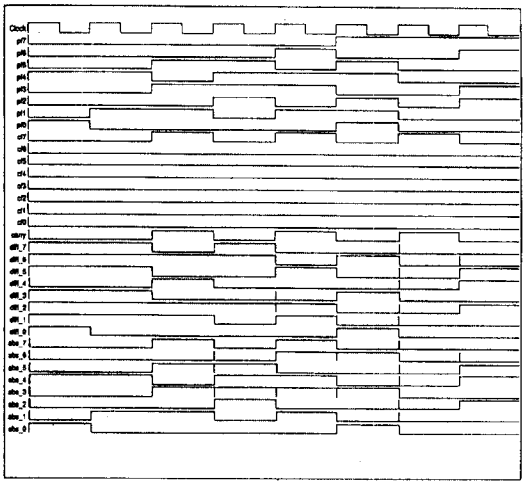


그림 10. 절대값 계산회로의 로직 시뮬레이션 결과
Fig. 10. Logic simulation results of the absoulue error calculation circuit.

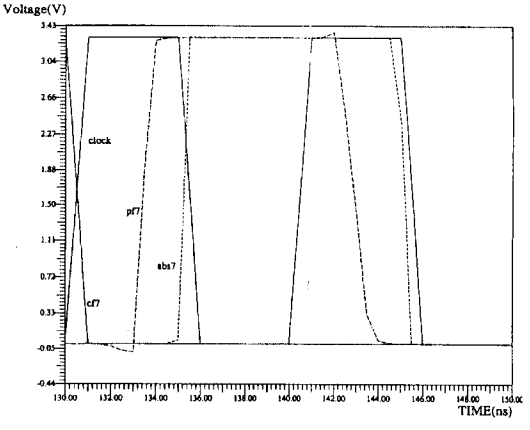


그림 11. PE의 stage I에 대한 SPICE 시뮬레이션 결과
Fig. 11. SPICE simulation results of stage I in a PE.

그림 10은 본 논문에서 설계된 움직임 추정기의 연산성능을 결정하는 절대값 계산부분에 대한 로직 시뮬레이션 결과인데 이 그림에서 pf, cf는 각각 PF, CF 데이터를 나타내고 diff는 CF와 PF 데이터의 차이이며, abs는 diff의 절대값을 나타낸다. 그림 11은 gate length가 0.6um 일 때 이 부분의 HSPICE 시뮬레이션 결과중 클럭과 CF, PF, 절대값 데이터의 MSB를 나타낸 것인데, 클럭으로부터 약 4ns 후에 절대값 데이터가 나옴을 보여준다. 이 결과를 표 2에 적용하면, 초

당 약 2.4 x 10^8 개의 움직임벡터를 계산해낼 수 있다. 그림 12에 칩 전체의 layout을 보였다. 이 칩의 설계에 사용된 트랜지스터는 약 389,000 개이고, 0.6um CMOS 공정인 경우 칩의 크기는 6.08mm x 6.49mm = 39.5 mm^2 이 된다.

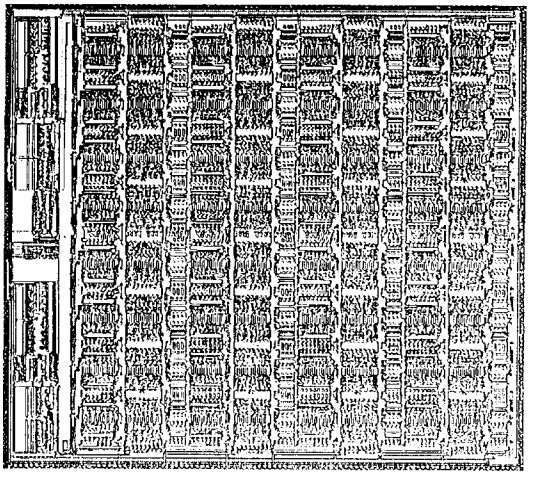


그림 12. 칩 layout
Fig. 12. Chip layout.

VI. 결 론

여러 가지 블록정합 알고리듬 중에서 가장 높은 정확도를 가지는 FBMA를 사용하면서도 고속 동작이 가능한 새로운 움직임 추정기가 제안되었다. 제안된 움직임 추정기는 간단하고, 느린 전송시간을 갖는 데이터 입출력 구조로도 PE 내부에서는 빠른 연산속도를 이룰 수 있는 매우 효과적인 data flow방식을 사용한다. 이 아키텍쳐는 연속되는 움직임벡터 연산간의 휴지시간을 제거해 100% PE 이용도로 throughput을 극대화하고, 적은 칩 핀 수를 갖도록 embedded memory 구조를 가진다. 또한, PE 수의 선택을 통한 성능 및 하드웨어 양의 조절이 매우 용이하여, MPEG의 다양한 분야에 적용 가능하고, 더 높은 정확도를 갖기 위한 fractional pel 단위의 움직임벡터를 계산하는 움직임 추정기의 설계에도 쉽게 응용될 수 있다.

참 고 문 헌

[1] F. May, "Full motion 64 kbit/s video codec with 8 DSP's," *Int. Workshop on 64 kbit/s Coding of Moving video,* Hanover, FRG, June 1988.

[2] T. Wehberg, H. Volkers, "Architecture for a programmable real time processor for digital video signals adapted to motion estimation algorithms," *SPIE,* vol. 1001 (Visual Communications and Image Processing 188), Cambridge, MA, pp. 908-916, Nov. 1988.

[3] M. Yamashina, T. Enomoto, T. Kunio, I. Tamitani, H. Harasaki, Y. Endo, T. Nishitani, M. Sata, K. Kikuchi, "A microprogrammable Real Time Video Signal Processor(VSP) for Motion compensation," *IEEE Journal of Solid-State Circuits,* vol. 23, no. 4, Aug. 1988.

[4] L. D. Vos, M. Stegherr, "Parameterizable VLSI Architecture for Full search Block-Matching Algorithm," *IEEE Trans. on Circuits and Systems,* vol. 36, no. 10, Oct. 1989.

[5] K. M. Kang, M. T. Sun, L. Wu, "A Family of VLSI Design for the Motion compensation Block-Matching Algorithm," *IEEE Trans. on Circuits and Systems,* vol. 36, no. 10 Oct. 1989.

[6] C. H. Hsiech, T. P. Lin, "VLSI Architecture for Block-Matching Motion Estimation Algorithm," *IEEE Trans. on Circuits and Systems for Video technology,* vol. 2, no. 2, June 1992.

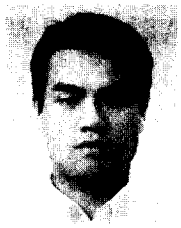
[7] N. H. E. Weste, K. Eshiraghan, *PRINCIPLES OF CMOS VLSI DESIGN,* ADDISON-WESLEY, Ch. 8, 1993.

[8] 김 준호, 유 재희, " Fractional pel 단위의 정확도를 처리 가능한 움직임 보상 프로세서의 VLSI 아키텍쳐 설계, " *대한 전자공학회 추계 종합 학술대회 논문집,* pp. 1398-1402, Nov. 1994

저 자 소 개

柳 在 熙(正會員)

1963年 3月 3日生. 1985年 2月 서울대학교 전자공학과 학사. 1987年 8月 Cornell 대학교 전기공학과 공학석사. 1990年 2月 Cornell 대학교 전기공학과 공학박사. 1990年 3月 ~ 1991年 3月 Texas Instruments, Dallas, VDL 연구원. 1996年 12月 현재 홍익대학교 전자공학과 조교수. 주관심분야는 Image, Speech signal processing VLSI 아키텍쳐 및 시스템 설계, DRAM, SRAM 회로설계 등임.



金 峻 虎(正會員)

1969年 7月 22日生. 1992年 2月 홍익대학교 전자공학과 학사. 1995年 2月 홍익대학교 전자공학과 석사. 1996年 12月 현재 현대전자 제2연구소 연구원. 주관심분야는 화상처리 VLSI 시스템 설계, DRAM 회로설계 등임.