Journal of Korea Multimedia Society Vol. 7, No. 8, August 2004(pp. 1170-1182)

3 단계 블록 매칭 알고리즘을 위한 4-경로 파이프라인 처리

정성태[†], 이상설[††], 남궁문[†††]

요 약

본 논문에서는 3단계 블록 매칭 알고리즘을 위한 새로운 4-경로 파이프라인 구조를 제안한다. 4-경로 파이프라인 구조를 위하여 현재 블록과 탐색 영역을 각각 4개의 부영역으로 분할하여 병렬처리하는 방법을 개발하였다. 4개의 부영역으로부터 메모리 접근의 충돌 없이 픽셀 데이터를 동시에 읽어 들이기 위한 메모리 분할 방법을 개발하였다. 제안된 구조는 C언어와 VHDL로 설계하여 시뮬레이션을 수행하였다. 실험 결과에 의하면 제안된 구조는 실시간 모션 추정 응용에 사용될 수 있는 높은 성능을 얻을 수 있었다.

A 4-way Pipelined Processing Architecture for Three-Step Search Block Matching Algorithm

Sung-Tae Jung[†], Sang-Seol Lee[††], Kung-Moon Nam[†††]

ABSTRACT

A novel 4-way pipelined processing architecture is presented for three-step search block-matching motion estimation. For the 4-way pipelined processing, we have developed a method which divides the current block and search area into 4 subregions respectively and processes them concurrently. Also, we have developed memory partitioning method to access pixel data from 4 subregions concurrently without memory conflict. The architecture has been designed and simulated with C language and VHDL. Experimental results show that the proposed architecture achieves a high performance for real time motion estimation.

Key words: Motion Estimation(모션 추정), Block Matching(블록 매칭), Pipeline(파이프라인)

1. 서 론

블록 매칭 알고리즘은 H.261과 MPEG과 같은 비디오 코딩 표준에 널리 이용되고 있다[1-3]. 블록 매칭 알고리즘은 동영상의 연속된 프레임 사이에 존재하는 높은 중복성을 이용하여 높은 압축률을 얻을 수 있도록 해준다. 가장 단순한 블록 매칭 알고리즘은 전탐색(full search) 알고리즘이다. 전탐색 알고리즘은 모든 후보 블록들과 비교를 수행함으로써 정확한 모션 벡터를 구할 수 있지만 많은 계산 복잡도를 가진다. 전탐색 알고리즘을 실시간으로 처리하기 위한 여러 가지 VLSI 구조들[4-8]이 제안되었지만 이들은 많은 량의 하드웨어를 필요로 한다. 전탐색 방법의 계산 복잡도를 줄이기 위한 방법으로 여러 가지 빠른 블록 매칭 알고리즘들[9-13]이 제안되었다. 이들 가운데, 3-단계 블록 매칭 알고리즘은 우수한 알고리즘으로 평가받고 있으며 CCITT RM8[2]과

※ 교신저자(Corresponding Author) : 이상설, 주소 : 전북 익산시 신용동 344-2(570-749), 전화 : 063)850-6889, FAX : 063)850-6889, E-mail : slee@wonkwang.ac.kr
접수일 : 2003년 11월 4일, 완료일 : 2003년 12월 31일
[†]종신회원, 원광대학교 전기전자및정보공학부 교수
(E-mail : stjung@wonkwang.ac.kr)
[††] 정회원, 원광대학교 전기전자및정보공학부 교수
[†††] 정회원, 원광대학교 토목환경·도시공학부 교수
(E-mail : ngmoon@wonkwang.ac.kr)
※ 본 연구는 한국과학재단 목적기초연구(R01-2000-000-00377 0)지원에 의해서 수행되었음.

MPEG SM3[3] 등에서 추천되고 있다. 3-단계 블록 매칭 알고리즘의 실시간 처리를 위하여 여러 가지 VLSI 구조들이 제안되었다[14-21].

Jheng은 참고 문헌[14]에서 시스톨릭 배열(Systolic Array)을 이용하여 3-단계 블록 매칭 알고리즘을 구현하는 방법을 제안하였다. 이 구조는 한 블록의 모션 벡터를 구하는 데에 3114 클럭 주기를 필요로 한다. 참고 문헌[15]에서는 처리기를 이진 트리 형태로 배열하는 구조를 제안하였다. 이 구조에서는 여러 가지 트리 형태를 제안하였는데, 8개의 메모리 모듈과 33개의 처리기를 사용하는 경우에 864 클럭 주기가 소요된다. Sheu[16]는 3개의 처리기를 파이프라인 방식을 연결하고 두 개의 메모리 모듈을 사용하는 구조를 제안하였다. 이 구조는 적은 수의 처리기와 메모리 모듈을 사용하여 하드웨어 비용을 줄였지만 한 블록의 모션 벡터 계산에 2993 클럭 주기를 필요로 한다. Jong[17,18]은 픽셀 데이터를 9개의 메모리 모듈에 효과적으로 분할하여 메모리 대역폭을 줄였으며 9개 처리기를 사용하여 794 클럭 주기에 한 블록에 대한 모션 벡터를 구했다. Costa[19]는 3개의 처리기를 사용하여 하드웨어 비용을 줄이는 방법을 제안하였으나 한 블록의 처리에 2976 클럭 주기를 필요로 한다. Dutt[20]는 3-단계 탐색뿐만 아니라 다른 알고리즘에도 적용 가능한 융통성 있는 구조를 제안하였는데, 3-단계 탐색을 사용하는 경우에 한 블록의 모션 벡터 계산에 1280 클럭 주기를 필요로 한다. Lai[21]는 Jong[17,18]의 방법을 바탕으로 하여 효율적인 데이터 링 구조를 사용함으로써 메모리 대역폭을 줄였으며 메모리 접근 회로를 단순화하였다. 이 방법에서는 한 블록의 모션 벡터 계산에 783 클럭 주기를 필요로 한다. Jong[17,18]과 Lai[21] 방법에서는 메모리 대역폭을 줄였지만 칩 내부의 지역 메모리를 필요로 한다.

본 논문에서는 3-단계 탐색 알고리즘을 효율적으로 구현할 수 있는 4-경로 파이프라인 처리 구조를 제안한다. 4-경로 파이프라인 처리 구조에서는 블록의 영역을 네 영역으로 분할하여 파이프라인에 의하여 병렬 처리한다. 네 영역을 병렬로 처리하기 위해서는 적절한 데이터를 파이프라인의 네 경로에 공급해 주어야 하는데, 메모리 충돌을 일어나지 않도록 하기 위하여 영상의 픽셀을 메모리에 분할 저장하고 픽셀을 접근하는 순서를 제어하는 방법을 개발하였다. 본 논문에서 제안된 구조는 8개의 메모리 모듈과 9개의 처리기를 사용하여 한 블록의 모션 벡터 계산에 341 클럭 주기를 필요로 한다. 제안된 방법은 기존의 방법에 비하여 한 블록의 모션 벡터 계산에 필요한 클럭 주기를 줄였다.

2. 알고리즘 변환

기존의 3-단계 탐색 블록 매칭 알고리즘의 각 단계 m에서는 현재 블록과 가장 비슷한 블록을 찾기 위하여 식 (1)과 같이 각 후보 위치 (i,j)에 대하여 후보 블록과 현재 블록 사이의 픽셀 값의 절대 차이의 합(SAD: Sum of Absolute Difference)을 구한다.

$$SAD^m(i,j)=\sum_{y=0}^{N-1}\sum_{x=0}^{N-1}|a(x,y)-b(x+d_x^m(i,j),y+d_y^m(i,j))| \quad (1)$$

여기에서 $a(x,y)$는 크기가 $N\times N$인 현재 블록의 픽셀 값이다. 본 논문에서는 현재 블록의 크기를 16×16으로 가정한다. $b(x+d_x^m(i,j),y+d_y^m(i,j))$는 탐색 영역의 픽셀 값을 나타낸다. $d_x^m(i,j)$와 $d_y^m(i,j)$는 각 단계 m에서 현재 블록과 후보 위치 (i,j) 사이의 누적된 변위를 나타내는 것으로서 $d_x^m(i,j)=d_x^{m-1}(i,j)+i\times\delta_m$, $d_y^m(i,j)=d_y^{m-1}(i,j)+j\times\delta_m$과 같이 정의된다. 이들의 초기값은 $d_x^0(i,j)=0$, $d_y^0(i,j)=0$이다. (i,j)는 (-1,-1), (-1, 0), (-1,1), (0,-1), (0,0), (0,1), (1,-1), (1,0), (1,1)과 같은 값을 가지며 9개의 후보 위치를 나타낸다. δ_m은 각 단계 m에서 각 후보 위치 사이의 거리이다($\delta_1=4$, $\delta_2=2$, $\delta_3=1$).

본 논문에서는 모션 벡터의 범위를 (-7,-7)에서 (7,7)까지로 제한한다. 그림 1에는 3단계 탐색 블록 매칭 과정의 한 예가 나타나 있다. 첫 단계에서는 가로 세로 방향으로 4씩 떨어져 있는 9개의 위치에서 SAD를 계산한 다음에 최소의 SAD 값을 가진 위치를 선택한다. 두 번째 단계에서는 첫 번째 단계에서 선택된 위치를 중심으로 9개의 후보 위치를 설정하고 SAD 값을 계산한다. 이때에는 후보 위치 사이의 거리가 2가 된다. 마찬가지로 세 번째 단계에서도 두 번째 단계에서 최소의 SAD를 갖는 위치를 중심으로 9개의 후보 위치에서 SAD를 계산한다. 이때에는 후보 위치 사이의 거리가 1이 된다. 3단계에서 최소의 SAD 값을 가지는 후보 위치와 현재 블록 사이의 변

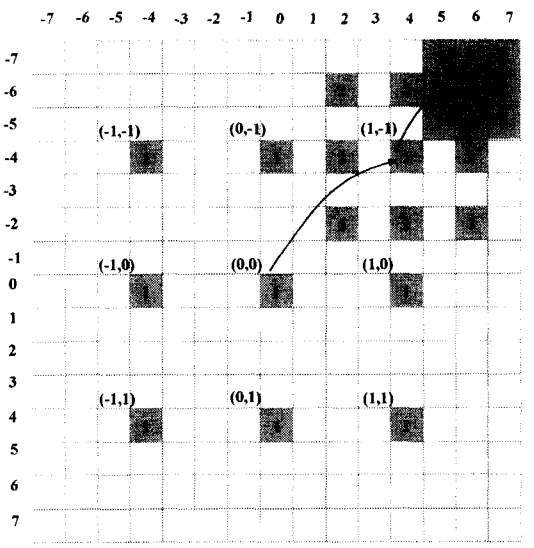
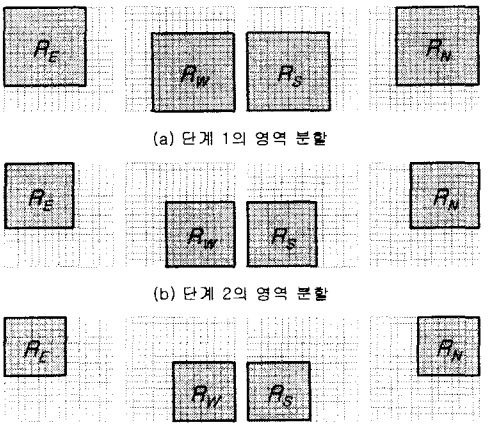


그림 1. 3-단계 탐색 블록 매칭 예

위가 최종 모션 벡터 값이 된다. 그림 1에서는 각 단계의 후보 위치에 단계 번호 1,2,3을 표시하였다. 그리고 1 단계의 후보 위치에는 위치 표시자 (i,j)를 좌측 상단에 표시하였다. 그림에서 화살표는 후보 위치 (0,0)의 이동 경로를 표시하고 있다.

본 논문에서는 현재 블록과 탐색 영역을 각각 4개의 부영역 R_E, R_W, R_S, R_N으로 분할하고 각 부영역을 병렬로 처리하는 방법을 제안한다. 그림 2에는 현재 블록에 대하여 4개의 부영역에 대한 분할 방법이 나타나 있다. 그림에 나타나 있듯이 4개의 부영역이 서로 겹치는데, 이것은 파이프라인 처리를 가능하게 하기 위해서이다. 각 단계 별로 분할 영역의 크기가 동적으로 변화하는데, 이것은 단계별로 탐색 영역의 크기가 달라지기 때문이다. 각 부영역의 크기는 단계 1에서는 12×12이고, 단계 2에서는 10×10, 단계 3에서는 9×9이다.



(a) 단계 1의 영역 분할

(b) 단계 2의 영역 분할

(c) 단계 3의 영역 분할

그림 2. 현재 블록에 대한 영역 분할

그림 3에는 탐색 영역에 대한 영역 분할 방법이 나타나있다. 탐색 영역은 동일한 크기를 갖는 4개의 부영역으로 분할된다. 각 단계별로 탐색 영역의 크기가 변하는데, 단계 1에서는 탐색 영역 크기가 24×24이고, 단계 2에서는 20×20이며, 단계 3에서는 18×18이다. 각 단계에서 부영역의 크기는 현재 블록의 부영역 크기와 동일하다.

병렬 처리를 위하여 식 (1)은 식 (2)와 같이 변환될 수 있다.

$$SAD^m(i,j) = SAD^m_{R_E}(i,j) + SAD^m_{R_W}(i,j) + SAD^m_{R_S}(i,j) + SAD^m_{R_N}(i,j) \quad (2)$$

여기에서, $SAD^m_{R_K}$는 부영역 R_K에서 현재 블록과 후보 블록 사이의 SAD 값을 의미하는 것으로서 식 (3)과 같이 정의된다.

$$SAD^m_{R_K}(i,j) = \sum_{f_y=0}^{f_y} \sum_{f_x=0}^{f_x} \sum_{n_y=0}^{n_y} \sum_{n_x=0}^{n_x} AD^m_{R_K}(i,j) \times u^m_{R_K}(i,j) \quad (3)$$

$$AD^m_{R_K}(i,j) = | a(x_{R_K}, y_{R_K}) - b(x_{R_K} + d^m_x(i,j), y_{R_K} + d^m_y(i,j)) |,$$

$$x_{R_E} = \delta_m \times n_x + f_x,\ y_{R_E} = \delta_m \times n_y + f_y,,$$

$$x_{R_N} = (N-1) - (\delta_m \times n_y + f_y),\ y_{R_N} = \delta_m \times n_x + f_x,,$$

$$x_{R_S} = \delta_m \times n_y + f_y,\ y_{R_S} = (N-1) - (\delta_m \times n_x + f_x),,$$

$$x_{R_W} = (N-1) - (\delta_m \times n_x + f_x),\ y_{R_W} = (N-1) - (\delta_m \times n_y + f_y),,$$

$$u^m_{R_E}(i,j) = f(\delta_m \times (i-2) + x_{R_E} < 0 \text{ and } \delta_m \times (j-2) + y_{R_E} < 0),$$

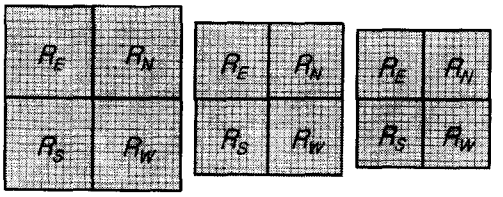
$$u^m_{R_N}(i,j) = f(\delta_m \times (i-2) + x_{R_N} \geq 0 \text{ and } \delta_m \times (j-2) + y_{R_N} < 0),$$

$$u^m_{R_S}(i,j) = f(\delta_m \times (i-2) + x_{R_S} < 0 \text{ and } \delta_m \times (j-2) + y_{R_S} \geq 0),$$

$$u^m_{R_W}(i,j) = f(\delta_m \times (i-2) + x_{R_W} \geq 0 \text{ and } \delta_m \times (j-2) + y_{R_W} \geq 0),$$

$$f(t) = \begin{cases} 1 & t = TRUE \\ 0 & t = FALSE \end{cases}$$

식 (3)에서 f_y, f_x, n_y, n_x는 부영역 픽셀들의 샘플링 순서를 제어한다. n_y와 n_x는 $\delta_m \times \delta_m$ 크기를 갖는 샘플 영역을 나타낸다. n_y와 n_x는 0에서 n_m 사이의 값을 가지는데, $n_1=2$, $n_2=4$, $n_3=8$ 이다. f_y와 f_x는 샘플 영역 내에서의 픽셀의 인덱스를 나타내고 0에서 f_m 사이의 값을 가지는데, $f_1=3$, $f_2=1$, $f_3=0$이다. $AD^m_{R_K}(i,j)$는 f_y, f_x, n_y, n_x 값



(a) 단계 1 (b) 단계 2 (c) 단계 3

그림 3. 탐색 영역에 대한 영역 분할

을 이용하여 샘플링된 현재 블록의 픽셀 $a(x_{R_K}, y_{R_K})$와 후보 블록 픽셀 $b(x_{R_K}+d_x^m(i,j)$, $y_{R_K}+d_y^m(i,j))$ 사이의 절대 차이 값을 나타낸다. 그런데, 앞에서 설명한 바와 같이 4개의 부영역이 서로 겹치는 부분이 있기 때문에 $AD_{R_K}^m(i,j)$ 값도 중복 계산된다. 중복된 부분에서 $AD_{R_K}^m(i,j)$ 값이 SAD 계산에 한번만 더해지도록 하기 위해서 $u_{R_K}^m(i,j)$ 가 사용된다.

그림 4에는 단계 1에서 현재 블록의 4개의 부영역에 대하여 f_y, f_x, n_y, n_x의 값과 그에 따라 접근되는 픽셀 (x_{R_K}, y_{R_K}) 사이의 관계가 나타나 있다. 픽셀 중에서 처음에 접근되는 9개 픽셀에는 순서대로 번호가 붙여져 있다.

그림 5에는 단계 2와 단계 3에서 부영역 R_E에 대한 f_y, f_x, n_y, n_x와 (x_{R_K}, y_{R_K}) 사이의 관계가 나타나 있다. 나머지 부영역에 대한 관계는 그림 4와 같은

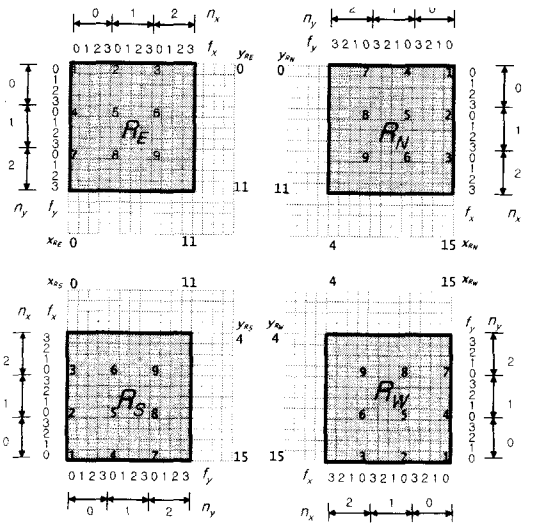


그림 4. 단계1에서 f_y, f_x, n_y, n_x와 (x_{R_K}, y_{R_K}) 사이의 관계

대칭적인 형태를 갖는다. 그림 5에서는 접근되는 픽셀들을 순서대로 20개까지 번호로 표시하였다.

9개의 각 후보 위치 (i,j)에 대한 $SAD_{R_K}^m(i,j)$ 값의 계산에서는 부영역의 픽셀들이 중복되지 않도록 해야 한다. 이를 위해서 $u_{R_K}^m(i,j)$ 가 사용되는데, 식 (3)에서는 i와 j값에 따라 현재 블록에서 각 부영역별로 $u_{R_K}^m(i,j)$ 값이 1이 되는 부분을 정의하고 있다. 이를 도식화하면 그림 6과 같다.

본 논문에서는 4개의 부영역에 대한 $SAD_{R_K}^m(i,j)$ 계산을 픽셀값 접근에 대한 충돌없이 병렬적으로 수행할 수 있는 구조를 제안한다.

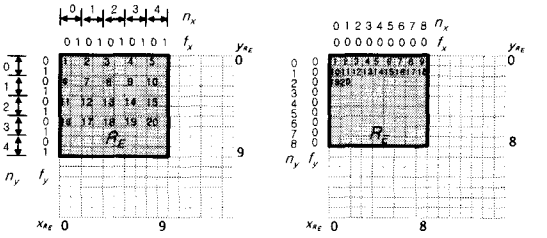


그림 5. 단계2와 단계 3에서 f_y, f_x, n_y, n_x와 (x_{R_K}, y_{R_K}) 사이의 관계

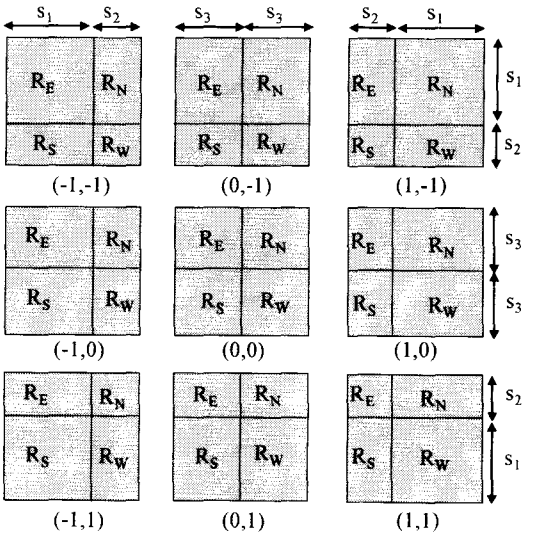


그림 6. $u_{R_K}^m(i,j)$ 함수의 도식적 표현

3. VLSI 구조

3.1 PE 내부 구조 및 연결 구조

본 논문에서 제안된 4-경로 파이프라인 처리 구조

는 9개의 처리기(PE : Processing Element)로 구성된다. 한 PE의 내부 구조는 그림 7과 같다. PE에는 절대 차이값 연산기(AD_{R_K}) 4개, 덧셈기 1개, 누산기 1개, 파이프라인 경로 4개, 방송 경로 4개가 포함되어 있다. 현재 블록의 픽셀 $a(x_{R_K}, y_{R_K})$와 후보 블록의 픽셀 $b(x_{R_K}+d_x^m(i,j)\ ,\ y_{R_K}+d_y^m(i,j))$가 각각 a_{R_K}와 b_{R_K}에 도착하면 AD_{R_K}의 연산이 실행되어 두 픽셀 사이의 절대 차이 값을 구한다. 4개의 부영역에 대한 AD_{R_K} 연산은 병렬로 수행되며 그 결과 값은 덧셈기에 의해 더해진 다음에 누산기에 의해 계속해서 누적된다. 후보 위치 $1(i,j)$에 대한 $SAD^m(i,j)$ 값은 하나의 처리기 $PE_{i,j}$에 의해 계산된다. 따라서 각 단계의 SAD 연산이 종료되면 $PE_{i,j}$의 누산기에는 후보 위치 (i,j)에 대한 SAD 값이 저장되고 이 값들을 비교하여 최소 값을 갖는 후보 위치를 찾은 다음에 다음 단계의 탐색 영역이 결정된다.

PE들은 그림 8과 같이 현재 블록의 픽셀 데이터를 전달하는 4 개의 파이프라인 경로와 후보 블록의 픽셀 데이터를 전달하는 4개의 방송(broadcasting) 경로에 의해 연결된다. 그림에서 a_{R_K}는 현재 블록의 픽셀 데이터를 나타내고 b_{R_K}는 후보 블록의 픽셀 데이터를 나타낸다. 부영역 R_E, R_W, R_S, R_N의 픽셀 데이터는 각각 $PE_{-1,-1}$, $PE_{1,1}$, $PE_{-1,1}$, $PE_{1,-1}$에서 시작하는 파이프라인에 입력된다.

3.2 데이터 흐름

그림 8에서 PDU(Programmable Delay Unit)는

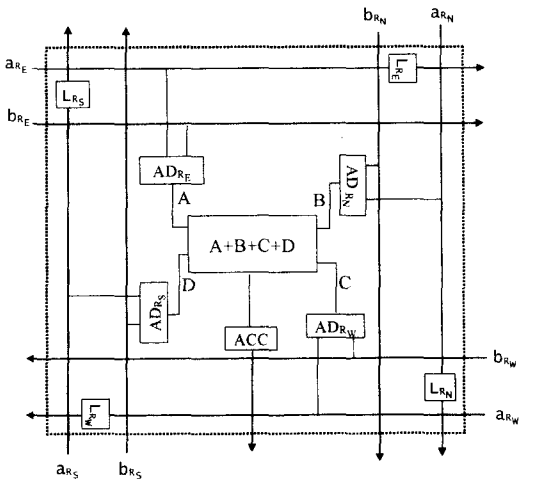


그림 7. PE의 내부 구조

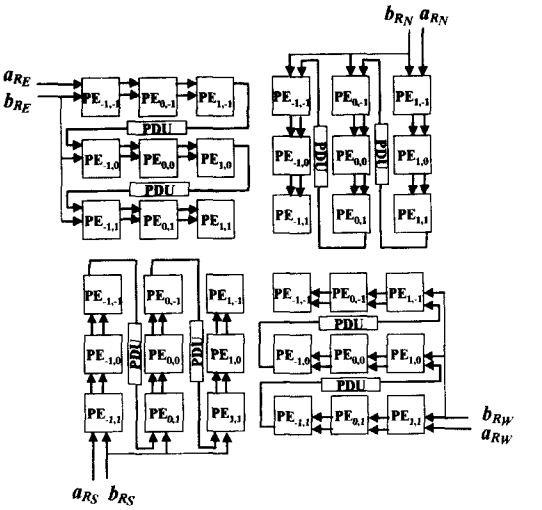


그림 8. PE의 연결 구조

각 단계별로 서로 다른 지연 시간을 갖는 FIFO(First In First Out) 버퍼이다. 단계 1의 지연시간은 0, 단계 2는 5, 단계3은 6이다. 표 1에는 제안된 구조에서 단계 1의 데이터 흐름이 나타나 있다.

표 1에는 $t=1$부터 각 시간별로 4개의 부영역으로부터 각 PE에 입력된 후보 영역의 픽셀과 현재 블록의 픽셀이 나타나 있다. 후보 블록의 픽셀은 전체 PE에 동시에 전달되므로 "후보 영역" 열에 한 번만 표시하였다. 표 1에서 색이 칠해진 셀은 해당 PE에서 현재 블록 픽셀과 후보 블록 픽셀 사이의 AD 연산을 실행한다는 것을 나타낸다. 현재 블록의 픽셀이 표시되었지만 색이 칠해지지 않은 셀은 해당 PE에서 AD 연산은 수행하지 않고 입력된 현재 블록 픽셀 데이터를 파이프라인 경로를 통하여 다음 PE에 전달해준다는 것을 나타낸다. $t=1$부터 $t=9$까지의 9주기 형태가 16번 반복되어 전체적으로 144주기 만에 단계 1의 SAD 연산이 완료된다. 표 1에서 각 PE는 첫 9주기 동안에 16번의 AD연산을 수행하고 있으며 이러한 형태가 16번 반복되므로 총 256번의 AD 연산을 수행함을 알 수 있다.

표 2에는 부영역 R_E에 대한 단계2의 데이터 흐름이 나타나 있다. 여기에서는 지면 관계상 한 부영역만 나타내었는데, 나머지 부영역에 대한 데이터 흐름도 비슷한 형태를 갖는다. 단계 2에서는 25 주기의 형태가 4번 반복되어 전체 SAD 연산에 100주기가 소요된다. 표 2에는 PDU 열이 포함되어 있는데, 픽

표 1. 단계 1의 데이터 흐름 (C: 시간, R: 부영역, S: 후보 영역)

C	R	S	현재 블록 픽셀								
			$PE_{-1,-1}$	$PE_{0,-1}$	$PE_{1,-1}$	$PE_{-1,0}$	$PE_{0,0}$	$PE_{1,0}$	$PE_{-1,1}$	$PE_{0,1}$	$PE_{1,1}$
1	R_E	-4,-4	0,0								
	R_N	19,-4			15,0						
	R_S	-4,19							0,15		
	R_W	19,19									15,15
2	R_E	0,-4	4,0	0,0							
	R_N	19,0			15,4			15,0			
	R_S	-4,15				0,15			0,11		
	R_W	15,19								15,15	11,15
3	R_E	4,-4	8,0	4,0	0,0						
	R_N	19,4			15,8			15,4			15,0
	R_S	-4,11	0,15			0,11			0,7		
	R_W	11,19							15,15	11,15	7,15
4	R_E	-4,0	0,4	8,0	4,0	0,0					
	R_N	15,-4		15,0	11,0			15,8			15,4
	R_S	0,19	0,11			0,7			4,15	0,15	
	R_W	19,15						15,15	11,15	7,15	15,11
5	R_E	0,0	4,4	0,4	8,0	4,0	0,0				
	R_N	15,0		15,4	11,4		15,0	11,0			15,8
	R_S	0,15	0,7			4,15	0,15		4,11	0,11	
	R_W	15,15					15,15	11,15	7,15	15,11	11,11
6	R_E	4,0	8,4	4,4	0,4	8,0	4,0	0,0			
	R_N	15,4		15,8	11,8		15,4	11,4		15,0	11,0
	R_S	0,11	4,15	0,15		4,11	0,11		4,7	0,7	
	R_W	11,15				15,15	11,15	7,15	15,11	11,11	7,11
7	R_E	-4,4	0,8	8,4	4,4	0,4	8,0	4,0	0,0		
	R_N	11,-4	15,0	11,0	7,0		15,8	11,8		15,4	11,4
	R_S	4,19	4,11	0,11		4,7	0,7		8,15	4,15	0,15
	R_W	19,11			15,15	11,15	7,15	15,11	11,11	7,11	15,7
8	R_E	0,4	4,8	0,8	8,4	4,4	0,4	8,0	4,0	0,0	
	R_N	11,0	15,4	11,4	7,4	15,0	11,0	7,0		15,8	11,8
	R_S	4,15	4,7	0,7		8,15	4,15	0,15	8,11	4,11	0,11
	R_W	15,11		15,15	11,15	7,15	15,11	11,11	7,11	15,7	11,7
9	R_E	4,4	8,8	4,8	0,8	8,4	4,4	0,4	8,0	4,0	0,0
	R_N	11,4	15,8	11,8	7,8	15,4	11,4	7,4	15,0	11,0	7,0
	R_S	4,11	8,15	4,15	0,15	8,11	4,11	0,11	8,7	4,7	0,7
	R_W	11,11	15,15	11,15	7,15	15,11	11,11	7,11	15,7	11,7	7,7
...											
144	R_E	7,7	11,11	7,11	3,11	11,7	7,7	3,7	11,3	7,3	3,3
	R_N	8,7	12,11	8,11	4,11	12,7	8,7	4,7	12,3	8,3	4,3
	R_S	7,8	11,12	7,12	3,12	11,8	7,8	3,8	11,4	7,4	3,4
	R_W	8,8	12,12	8,12	4,12	12,8	8,8	4,8	12,4	8,4	4,4

셀은 PDU에 2 주기 동안 머무르고 다음 PE에 전달되고 있다.

표 3에는 부영역 R_E에 대한 단계3의 데이터 흐름

표 2. 부영역 RE에 대한 단계 2의 데이터 흐름(C: 시간, R: 부영역)

C	R	현재 블록 픽셀									
		$PE_{-1,-1}$	$PE_{0,-1}$	$PE_{1,-1}$	PDU	$PE_{-1,0}$	$PE_{0,0}$	$PE_{1,0}$	PDU	$PE_{-1,1}$	$PE_{0,1}$	$PE_{1,1}$
1	-2,-2	0,0										
2	0,-2	2,0	0,0									
3	2,-2	4,0	2,0	0,0								
4	4,-2	6,0	4,0	2,0	0,0							
5	6,-2	8,0	6,0	4,0	2,0	0,0						
6	-2,0	0,2	8,0	6,0	4,0	2,0	0,0					
7	0,0	2,2	0,2	8,0	6,0	4,0	2,0	0,0				
8	2,0	4,2	2,2	0,2	8,0	6,0	4,0	2,0	0,0			
9	4,0	6,2	4,2	2,2	0,2	8,0	6,0	4,0	2,0	0,0		
10	6,0	8,2	6,2	4,2	2,2	0,2	8,0	6,0	4,0	2,0	0,0	
11	-2,2	0,4	8,2	6,2	4,2	2,2	0,2	8,0	6,0	4,0	2,0	0,0
12	0,2	2,4	0,4	8,2	6,2	4,2	2,2	0,2	8,0	6,0	4,0	2,0
13	2,2	4,4	2,4	0,4	8,2	6,2	4,2	2,2	0,2	8,0	6,0	4,0
14	4,2	6,4	4,4	2,4	0,4	8,2	6,2	4,2	2,2	0,2	8,0	6,0
15	6,2	8,4	6,4	4,4	2,4	0,4	8,2	6,2	4,2	2,2	0,2	8,0
16	-2,4	0,6	8,4	6,4	4,4	2,4	0,4	8,2	6,2	4,2	2,2	0,2
17	0,4	2,6	0,6	8,4	6,4	4,4	2,4	0,4	8,2	6,2	4,2	2,2
18	2,4	4,6	2,6	0,6	8,4	6,4	4,4	2,4	0,4	8,2	6,2	4,2
19	4,4	6,6	4,6	2,6	0,6	8,4	6,4	4,4	2,4	0,4	8,2	6,2
20	6,4	8,6	6,6	4,6	2,6	0,6	8,4	6,4	4,4	2,4	0,4	8,2
21	-2,6	0,8	8,6	6,6	4,6	2,6	0,6	8,4	6,4	4,4	2,4	0,4
22	0,6	2,8	0,8	8,6	6,6	4,6	2,6	0,6	8,4	6,4	4,4	2,4
23	2,6	4,8	2,8	0,8	8,6	6,6	4,6	2,6	0,6	8,4	6,4	4,4
24	4,6	6,8	4,8	2,8	0,8	8,6	6,6	4,6	2,6	0,6	8,4	6,4
25	6,6	8,8	6,8	4,8	2,8	0,8	8,6	6,6	4,6	2,6	0,6	8,4
...												
96	-1,7	1,9	9,7	7,7	5,7	3,7	1,7	9,5	7,5	5,5	3,5	1,5	9,3	7,3
97	1,7	3,9	1,9	9,7	7,7	5,7	3,7	1,7	9,5	7,5	5,5	3,5	1,5	9,3
98	3,7	5,9	3,9	1,9	9,7	7,7	5,7	3,7	1,7	9,5	7,5	5,5	3,5	1,5
99	5,7	7,9	5,9	3,9	1,9	9,7	7,7	5,7	3,7	1,7	9,5	7,5	5,5	3,5
100	7,7	9,9	7,9	5,9	3,9	1,9	9,7	7,7	5,7	3,7	1,7	9,5	7,5	5,5

이 나타나 있다. 나머지 부영역에 대한 데이터 흐름도 비슷한 형태를 갖는다. 단계 3에서는 전체 *SAD* 연산에 81주기가 소요된다. 표에서 $t=37$부터 $t=72$까지의 주기는 생략되었는데, 이 부분에서는 $t=28$부터 $t=36$ 사이의 9주기의 형태가 4회 반복된다. 표 3의 PDU 에서는 픽셀이 6 주기 동안 머무르고 다음 PE에 전달되고 있다.

3.3 메모리 분할 및 접근

제안된 구조의 각 주기에는 현재 블록의 4 부영역으로부터 4개의 픽셀 데이터와 후보 영역의 4 부영역

표 3. 부영역 R_E에 대한 단계 3의 데이터 흐름

C	S	현재 블록 픽셀																						
		PE	PE	PE	PDU							PE	PE	PE	PDU							PE	PE	PE
1	-1,-1	0,0																						
2	0,-1	1,0	0,0																					
3	1,-1	2,0	1,0	0,0																				
4	2,-1	3,0	2,0	1,0	0,0																			
5	3,-1	4,0	3,0	2,0	1,0	0,0																		
6	4,-1	5,0	4,0	3,0	2,0	1,0	0,0																	
7	5,-1	6,0	5,0	4,0	3,0	2,0	1,0	0,0																
8	6,-1	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0															
9	7,-1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0														
10	-1,0	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0													
11	0,0	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0												
12	1,0	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0											
13	2,0	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0										
14	3,0	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0									
15	4,0	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0								
16	5,0	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0							
17	6,0	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0						
18	7,0	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0					
19	-1,1	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0				
20	0,1	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0			
21	1,1	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0	0,0		
22	2,1	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0	1,0		
23	3,1	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0	2,0		
24	4,1	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0	3,0		
25	5,1	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0	4,0		
26	6,1	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0	5,0		
27	7,1	8,2	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0	6,0		
28	-1,2	0,3	8,2	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0	7,0		
29	0,2	1,3	0,3	8,2	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1	8,0		
30	1,2	2,3	1,3	0,3	8,2	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1	0,1		
31	2,2	3,3	2,3	1,3	0,3	8,2	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1	1,1		
32	3,2	4,3	3,3	2,3	1,3	0,3	8,2	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1	2,1		
33	4,2	5,3	4,3	3,3	2,3	1,3	0,3	8,2	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1	3,1		
34	5,2	6,3	5,3	4,3	3,3	2,3	1,3	0,3	8,2	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1	4,1		
35	6,2	7,3	6,3	5,3	4,3	3,3	2,3	1,3	0,3	8,2	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1	5,1		
36	7,2	8,3	7,3	6,3	5,3	4,3	3,3	2,3	1,3	0,3	8,2	7,2	6,2	5,2	4,2	3,2	2,2	1,2	0,2	8,1	7,1	6,1		
...																								
73	-1,7	0,8	8,7	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	8,6	7,6	6,6	5,6	4,6	3,6	2,6	1,6	0,6	8,5	7,5
74	0,7	1,8	0,8	8,7	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	8,6	7,6	6,6	5,6	4,6	3,6	2,6	1,6	0,6	8,5
75	1,7	2,8	1,8	0,8	8,7	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	8,6	7,6	6,6	5,6	4,6	3,6	2,6	1,6	0,6
76	2,7	3,8	2,8	1,8	0,8	8,7	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	8,6	7,6	6,6	5,6	4,6	3,6	2,6	1,6
77	3,7	4,8	3,8	2,8	1,8	0,8	8,7	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	8,6	7,6	6,6	5,6	4,6	3,6	2,6
78	4,7	5,8	4,8	3,8	2,8	1,8	0,8	8,7	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	8,6	7,6	6,6	5,6	4,6	3,6
79	5,7	6,8	5,8	4,8	3,8	2,8	1,8	0,8	8,7	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	8,6	7,6	6,6	5,6	4,6
80	6,7	7,8	6,8	5,8	4,8	3,8	2,8	1,8	0,8	8,7	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	8,6	7,6	6,6	5,6
81	7,7	8,8	7,8	6,8	5,8	4,8	3,8	2,8	1,8	0,8	8,7	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	8,6	7,6	6,6

으로부터 4개의 픽셀 데이터를 메모리 접근에 대한 충돌 없이 동시에 읽어 들여야 한다. 이를 위하여 본 논문의 구조에서는 현재 블록의 픽셀 데이터를 그림 9와 같은 형태로 4개의 메모리 모듈에 나누어 저장한다. 각 픽셀에 표시되어 있는 번호는 픽셀이 저장되어 있는 메모리 모듈 번호를 나타낸다. 표 1의 데이터 흐름을 살펴보면 $t=1$의 주기에서 동시에 접근되어야 할 픽셀은 $a(0,0)$, $a(15,0)$, $a(0,15)$, $a(15,15)$이다. 이들 픽셀은 각각 메모리 모듈 1, 2, 3, 4에 나뉘어 저장되어 있으므로 메모리 접근 충돌 없이 동시에 접근될 수 있다.

후보 영역의 픽셀들도 같은 방법으로 4개의 메모리 모듈에 나뉘어 저장된다. 후보 영역과 현재 블록은 그림 10 (a)와 같이 가로 세로 방향으로 4픽셀씩 떨어져 있다. 단계 2에서는 그림 10 (b)와 같이 단계 1에서 최소 *SAD*를 갖는 후보 위치를 중심으로 2픽셀씩 떨어져 있다. 단계 3에서는 그림 10 (c)와 같이 단계 2에서 최소 *SAD*를 갖는 후보 위치를 중심으로 1픽셀씩 떨어져 있다.

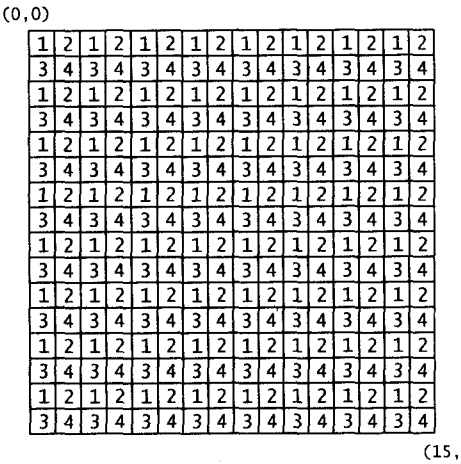
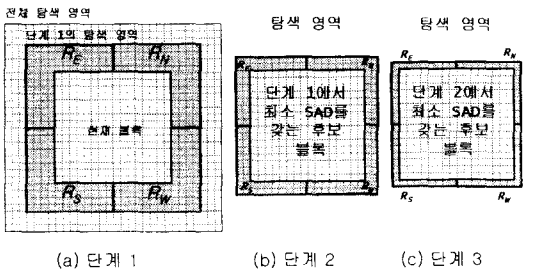


그림 9. 현재 블록의 메모리 분할 형태



(a) 단계 1 (b) 단계 2 (c) 단계 3

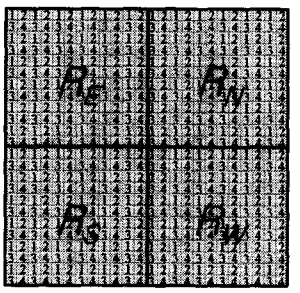
그림 10. 후보 영역의 위치

탐색 영역의 위치가 그림 10과 같으므로 탐색 영역의 각 부영역의 메모리 분할은 그림 11과 같이 된다. 단계 1과 단계2에서는 후보 영역의 픽셀이 저장되어 있는 메모리 모듈의 형태가 현재 블록과 동일하지만 단계 3에서는 다른 형태를 갖는다.

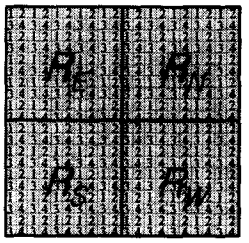
각 부영역의 픽셀 데이터를 접근하기 위해서는 각 메모리 모듈에 대한 주소 생성기와 메모리 모듈로부터 읽혀진 데이터를 해당 부영역에 전달하는 경로 배정자가 필요하다. 각 부영역의 픽셀을 어느 메모리 모듈에서 읽는지는 f_y, f_x, n_y, n_x 값을 이용하여 계산할 수 있다. 표 4에는 이들 값과 현재 블록에 대하여 접근되는 메모리 모듈과의 관계가 나타나 있다. 단계 1과 단계2에서는 n_y, n_x 값에는 무관하고 f_y, f_x 값이 홀수인지 짝수 인지에 따라서 각 부영역 별로 접근하는 메모리 모듈이 결정된다. 단계3에서는 f_y, f_x 값 대신에 n_y, n_x 값에 따라 결정된다.

표 5에는 f_y, f_x, n_y, n_x 값과 탐색 영역에 대하여 접근되는 메모리 모듈과의 관계가 나타나 있다. 단계 1과 단계2에서는 현재 블록의 경우와 동일하고 단계 3에서는 접근되는 메모리 모듈이 다르다.

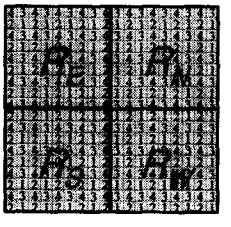
각 메모리 모듈에서의 픽셀 주소는 그림 12와 같이 (0,0)에서 (7,7)까지의 값을 가지며 이웃한 4개의 픽셀은 동일한 주소를 갖는다.



(a) 단계 1



(b) 단계 2



(c) 단계 3

그림 11. 탐색 영역의 메모리 분할

표 4. f_y, f_x, n_y, n_x 와 현재 블록 접근에 대한 메모리 모듈 사이의 관계

단계	f_y	f_x	n_y	n_x	R_E	R_N	R_S	R_W
1과 2	짝수	짝수	X	X	1	2	3	4
	짝수	홀수	X	X	2	4	1	3
	홀수	짝수	X	X	3	1	4	2
	홀수	홀수	X	X	4	3	2	1
3	X	X	짝수	짝수	1	2	3	4
	X	X	짝수	홀수	2	4	1	3
	X	X	홀수	짝수	3	1	4	2
	X	X	홀수	홀수	4	3	2	1

표 5. f_y, f_x, n_y, n_x와 탐색 영역 접근에 대한 메모리 모듈 사이의 관계

단계	f_y	f_x	n_y	n_x	R_E	R_N	R_S	R_W
1과 2	짝수	짝수	X	X	1	2	3	4
	짝수	홀수	X	X	2	4	1	3
	홀수	짝수	X	X	3	1	4	2
	홀수	홀수	X	X	4	3	2	1
3	X	X	짝수	짝수	4	3	2	1
	X	X	짝수	홀수	3	1	4	2
	X	X	홀수	짝수	2	4	1	3
	X	X	홀수	홀수	1	2	3	4

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)	(7,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	(7,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)	(7,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)	(7,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)	(7,4)
(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)	(7,5)
(0,6)	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)	(7,6)
(0,7)	(1,7)	(2,7)	(3,7)	(4,7)	(5,7)	(6,7)	(7,7)

그림 12. 메모리 모듈의 주소

식 (3)에는 f_y, f_x, n_y, n_x 값을 이용하여 현재 블록의 각 부영역의 픽셀 주소 (x_{R_K}, y_{R_K})를 구하는 식이 나타나 있다. 그런데, 식 (3)은 픽셀 주소가 (0,0)에서 (15,15)까지 인 현재 블록에 대한 식이므로, 이 식을 메모리 모듈에 적용하기 위해서는 2로 나누어

야 한다. 표 6에는 각 단계별로 각 부영역의 픽셀 데이터 접근을 위한 주소 계산식이 나타나 있다. 이들은 모두 식 (3)으로부터 유도한 것이다.

표 6. 각 부영역의 메모리 모듈 접근 주소

단계	각 부영역의 픽셀 주소	
1	$x_{R_E}=2\times n_x+f_x/2$	$y_{R_E}=2\times n_y+f_y/2$
	$x_{R_N}=7-(2\times n_y+f_y/2)$	$y_{R_N}=2\times n_x+f_x/2$
	$x_{R_S}=2\times n_y+f_y/2$	$y_{R_S}=7-(2\times n_x+f_x/2)$
	$x_{R_W}=7-(2\times n_x+f_x/2)$	$y_{R_W}=7-(2\times n_y+f_y/2)$
2	$x_{R_E}=n_x$	$y_{R_E}=n_y$
	$x_{R_N}=7-n_y$	$y_{R_N}=n_x$
	$x_{R_S}=n_y$	$y_{R_S}=7-n_x$
	$x_{R_W}=7-n_x$	$y_{R_W}=7-n_y$
3	$x_{R_E}=n_x/2$	$y_{R_E}=n_y/2$
	$x_{R_N}=7-n_y/2$	$y_{R_N}=n_x/2$
	$x_{R_S}=n_y/2$	$y_{R_S}=7-n_x/2$
	$x_{R_W}=7-n_x/2$	$y_{R_W}=7-n_y/2$

4. 실험 결과

본 논문에서 제안된 구조의 동작을 검증하기 위해서 먼저 C언어를 사용하여 시뮬레이션을 수행하였다. C언어를 사용하여 PE 내부 구조, PE 연결 구조, 주소 생성기, 데이터 경로 배정자, 메모리 모듈 등 전체 구조를 모두 구현하여 여러 연속된 이미지를 입력하여 블록 매칭을 수행하였다. C 언어를 사용한 시뮬레이션에서는 병렬 처리가 아닌 순차 처리 방법이 사용되었다. 즉, *SAD* 계산을 위한 모든 연산은 하나씩 순차적으로 수행되었다. 그러나, 이러한 시뮬레이션으로도 제안된 알고리즘이 올바르게 결과를 생성하는지는 검증할 수 있었다. 그림 13에는 연속된 두 프레임의 블록들에 대하여 시뮬레이션을 통하여 검출한 모션 벡터가 나타나 있다. 그림 13 (c)에서 각 사각형은 현재 블록을 나타내고 시각형 내부의 선들이 모션 벡터를 나타낸다. 시뮬레이션 프로그램은 CPU가 Pentium 4이고 운영체제가 윈도우즈 XP인 시스템에서 구현되고 실행되었다. 시뮬레이션 프로그램을 이용하여 100개의 프레임에 대하여 연속적으로 모션 벡터를 구하는 데에 34초가 소요되었다.

또한 본 논문에서는 하드웨어 설계 언어인 VHDL을 이용하여 제안된 구조를 설계하고 시뮬레이션을 수행하였다. 사용된 VHDL 개발 툴은 ModelSim, ActiveHDL, Xilinx ISE인데, ModelSim은 시뮬레이션에 사용했고, ActiveHDL은 계층적인 모형을 구성과 VHDL 프로그래밍에 사용했으며, Xilinx ISE은 합성과 FPGA용 다운로드 비트맵 데이터를 추출하는 과정에 사용했다. VHDL은 계층적인 모형으로 개발되고 시뮬레이션 되었다. 즉, 컴포넌트 사이의 연결에 대해서는 VHDL 코드를 사용하지 않고 블록도를 이용하여 기술하였으며 해당 VHDL 코드는 소프트웨어에 의해 자동으로 생성되었다. 그리고 최하위 블록에 대해서는 VHDL을 사용하여 그 행위를 기술하였다. 그림 14에는 제안된 전체 시스템 구조의 블럭도가 나타나 있다.

그림 14의 모듈 중에서 AVR, FIFO, 이미지 센서, 메모리 모듈은 외부 하드웨어 모듈이고 ThreeStepSearch 모듈과 Arbiter_Router_Interface 모듈이 VLSI로 구현되어야 할 모듈이다. 그림 15에는 ThreeStepSearch 모듈의 상세 설계가 나타나 있다. ThreeStepSearch 모듈은 인가되는 픽셀 데이터를 가지고 3 단계 탐색을 수행하는 모듈로서, TreeStepSearchControl 모듈과 SingleStepSearch

(a) 프레임 1 (b) 프레임 2 (c) 모션 벡터

그림 13. 연속된 프레임에 대한 모션 벡터 추출

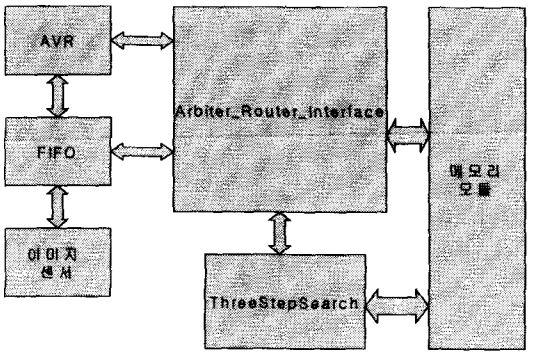


그림 14. 제안된 구조의 최상위 단계의 블록 구조

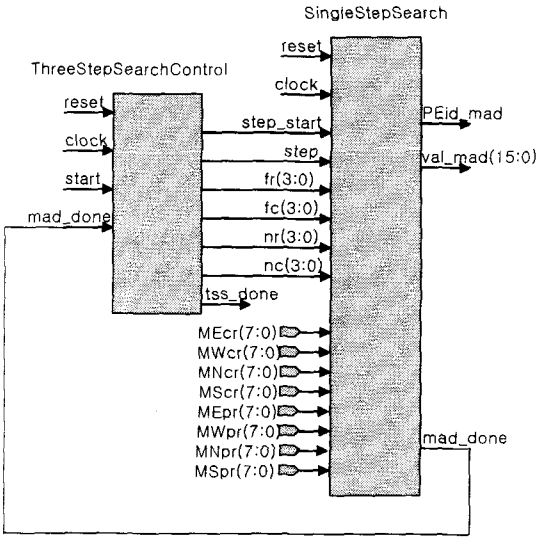


그림 15. ThreeStepSearch 모듈

모듈로 구성된다. TreeStepSearchControl 모듈은 VHDL 코드로 작성되는데, SingleStepSerch 모듈에게 3 단계에 걸쳐 각 단계마다 블록 매칭을 수행하도록 지시하고 블록 매칭이 끝나면 MAD를 구하게 하는 제어 신호를 제공한다.

그림 16의 시뮬레이션 결과는 블록 매칭의 세 번째 단계가 종료된 후에 MAD가 최소인 PE의 번호(PEid_mad)와 그 값(val_mad)이 얻어지는 과정을 보여준다. MAD 결정상태의 첫 번째와 두 번째 클럭에는 PE8을 마지막 세 번째 클럭에는 PE2를 최소의 MAD를 갖는 PE로 결정하게 되고 그 때 MAD 값은 10C0임을 알 수 있다. 최종 클럭에 최소 MAD가 결정되었음을 알리는 tss_done 신호가 함께 출력되는 것을 보여주고 있다.

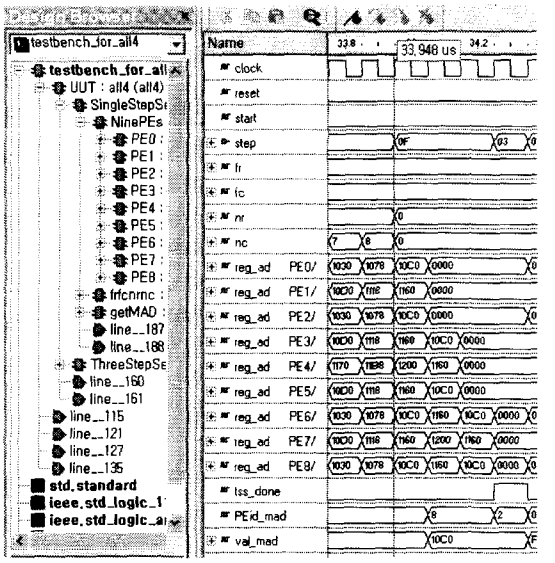


그림 16. 시뮬레이션 결과

Arbiter_Router_Interface 모듈은 Data_arbiter, Data_router, Memory_decoder, Address_generator 모듈로 구성되는데, 전체적인 제어 신호를 제공한다. Data_arbiter 모듈에서는 PE와 카메라 모듈 사이의 메모리에 대한 접근권을 제어하는 동작을 수행한다. Data_router 모듈은 8개의 메모리 모듈에서 읽혀진 픽셀 데이터를 PE에 전달하는 역할을 수행한다.이 모듈에서는 PE가 3단계 블록 매칭을 정확하게 수행하기 위해 동적으로 메모리 데이터와 파이프라인 입력 경로를 배정하는 동작을 VHDL로 코딩 구현하였다. Address_generator 모듈은 PE에 인가되어야 할 픽셀 데이터의 주소를 생성한다. 또한 이 모듈에서는 매 단계가 종료된 후 MAD로 결정된 PE에 따라 탐색 영역에 대한 다음 단계의 기본 주소를 계산한다. Memory_decoder 모듈은 8 개의 메모리 모듈에 제어 신호를 인가시킨다.

본 논문에서 설계된 블록 매칭 VLSI 구조는 VHDL 합성 도구인 "FPGA Express VHDL"을 이용해 Virtex2 XC2V1000-4bg575 FPGA에 매핑되었다. 매핑 결과 로직셀은 37%가 사용되었으며, I/O 블록은 92%가 사용되었다. 그리고 최대 시간지연은 9.573ns로 100MHz로 동작할 수 있었다. 그림 17에는 FPGA 매핑 결과가 나타나 있다.

3 단계 블록 매칭 알고리즘을 위해 제안된 기존의 구조와 본 논문에서 제안된 구조의 비교가 표 7에

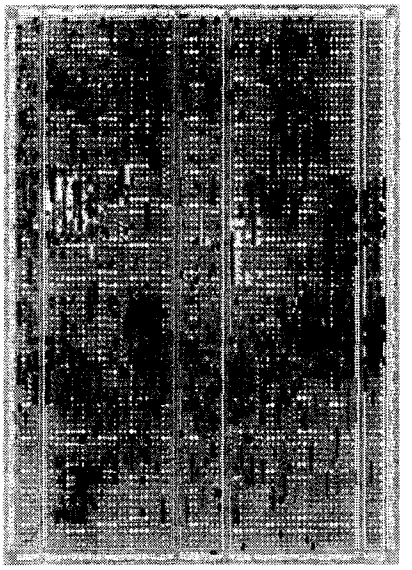


그림 17. FPGA 매핑 결과

표 7. 다른 구조와의 비교

	입력 데이터 포트(비트)	PE 수	클럭 주기	지역 메모리 (바이트)
제안된 구조	64	9	341	0
Jheng[14]	16	8	3114	0
Jheng[15]	64	33	864	0
Sheu[16]	16	3	2993	0
Jong[17,18]	16	9	794	1440
Costa[19]	16	3	2976	0
Dutta[20]	136	9	1280	0
Lai[21]	24	9	783	960

나타나 있다. 제안된 구조는 각 단계에서 SAD 값을 계산하는 데 $(f_m+1)^2\times(n_m+1)^2$ 클럭 주기를 필요로 하므로, 한 블록에 대한 SAD 계산에 $4^2\times3^2+2^2\times5^2+1^2\times9^2=325$클럭 주기가 소요된다. 그리고 MAD 계산과 다음 단계를 위한 탐색 영역 주소 계산에 16 클럭이 소요되어 한 블록의 모션 벡터 계산에 총 341 클럭이 소요된다. Jong[17,18]과 Lai[21]의 구조에 비교하여 본 논문에서 제안된 구조는 지역 메모리를 사용하지 않으므로 하드웨어를 적게 사용하고 있는 반면에 입력 데이터 포트는 더 많이 사용하고 있다.

블록의 크기가 $N\times N$, 영상의 크기가 $X\times Y$, 초당 프레임 수가 F일 때에 모션 벡터를 실시간으로 구하기 위한 초당 처리해야할 블록 수는 다음과 같이 계산할 수 있다.

$$B=\frac{X\times Y}{N\times N}\times F$$

위의 식에 의하면 H.261(352x288, 30 프레임/초를 가정) 응용을 실시간 처리하기 위해서는 초당 11.9K 개 블록의 모션 벡터를 구해야 하고, MPEG(720×480, 30 프레임/초를 가정) 응용을 위해서는 초당 40.5K개 블록의 모션 벡터를 구해야하며, HDTV (1920×1035, 30 프레임/초를 가정) 응용을 위해서는 초당 232.9K 개 블록의 모션 벡터를 구해야 한다.

한 블록의 모션 벡터 계산에 필요로 하는 클럭 주기가 C이고 초당 처리해야 할 블록수가 B인 응용을 처리하기 위해 필요로 하는 클럭 주파수 T는 다음 식과 같이 구할 수 있다.

$$T\geq B\times C$$

위 식에 의하여 본 논문에서 제안된 구조는 H.261 응용은 4.1MHz, MPEG 응용은 13.8MHz, HDTV 응용은 79.4MHz 이상의 클럭 속도를 필요로 한다. 그런데, 제안된 구조를 FPGA에 매핑시켜본 결과 100MHz의 동작 속도를 얻을 수 있다는 시뮬레이션 결과를 살펴 볼 때에 본 논문에서 제안한 구조는 HDTV 급의 고화질 영상에 대해서도 실시간으로 모션 벡터를 얻을 수 있음을 알 수 있다. 다른 논문에서 제안된 구조에서는 HDTV 급의 고화질 영상에 대해 실시간 모션 벡터를 얻기 위해서는 높은 클럭 속도를 사용해야 하거나 모션 벡터 감지기를 여러 개를 병렬로 사용하여 분할 처리해야 한다.

5. 결 론

본 논문에서는 3단계 블록 매칭 알고리즘을 수행하는 4-경로 파이프라인 구조를 제안하였다. 제안된 구조는 4-경로 파이프라인을 이용하여 처리 속도를 증가시킴으로써 큰 해상도의 비디오 응용에서도 실시간 모션 감지를 위해 사용될 수 있다. 본 논문에서는 4-경로 파이프라인 처리를 위하여 현재 블록과 탐색 영역을 4개의 부영역으로 분할하여 병렬 처리하는 방법을 고안하였고 4개의 부영역으로부터 픽셀 데이터를 메모리 충돌 없이 병렬로 읽어 들이기 위한 메모리 분할 방법을 고안하였다. 본 논문에서는 제안

된 구조를 C언어와 VHDL 언어를 이용하여 설계하고 시뮬레이션을 수행함으로써 동작성을 검증하였다.

참 고 문 헌

[1] CCITT SGXV, "Description of Reference Model 8 (RM8)", Document 525, Working Part XV/4, specialists Group on Coding for Visual Technology, 1989.

[2] MPEG, "ISO CD11172-2: Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5Mbits/s", Nov. 1991.

[3] W.Y. Zou, "Digital HDTV Compression Techniques", IEEE Transactions on Broadcasting, Vol. 37, No. 4, pp. 131-133,. 1991.

[4] J.-F. Shen, L.-G. Chen, H.-C. Chang, and T.-C. Wang, "Low power full-search block-matching motion estimation chip for H.263+," Proceedings of the International Symposium on Circuits and Systems, Vol. 4, pp. 299-302, 1999.

[5] K.-S. Kim, D.-J. Lee, H.-J. Yoo and K. Lee, "Low-power full-search motion estimator architecture suitable for random-block match," IEEE Asia Pacific Conference on ASICs, pp. 210-212, 1999.

[6] M.A. Elgamel, A.M. Shams, X. Xueling, and M.A. Bayoumi, "Enhanced low power motion estimation VLSI architectures for video compression," IEEE International Symposium on Circuits and Systems, Vol. 4, pp. 474-477, 2001.

[7] J.F. Lopez, F. Tobajas, S. Lopez, P. Cortes, S. Lalchand, and R. Sarmiento, "VLSI video processing elements for real time applications," Annual Conference of the Industrial Electronics Society(IECON 02), Vol. 3, pp. 1930-1935, 2002.

[8] N. Roma and L. Sousa, "Efficient and configurable full-search block-matching processors," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12, No. 12, pp. 1160-1167, 2002.

[9] T. Koga, K. Iinuma, A. Hirago, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe image coding for video conferencing", IEEE Nat. Telecommunication Conference, Vol. 4, pp. G5.3.1-G5.3.5, 1981.

[10] R. Li, B. Zeng and M.L. Liou, "A New Three-step Search Algorithm for Block Motion Estimation," IEEE Trans. on Circuits and Systems for Video Technology, vol. 4, no. 4, pp. 438-442, 1994.

[11] H. Nisar, T.-S. Choi, "An advanced center biased three step search algorithm for motion estimation", IEEE International Conference on Multimedia and Expo, Vol. pp. 95 -98, 2000.

[12] L. Po and W. Ma, "A novel four-step search algorithm for fast block motion estimation", IEEE Trans. on Circuits and Systems for Video Technology, Vol. 6, No. 3, pp. 313-317, 1996.

[13] C. W. Lin, Y.J. Chang, and Y. C. Chen, "Hierachical Motion Estimation Algorithm Based on Pyramidal Sucessive Elimination", International Computer Symposium, Proceedings of Workshop on Image Processing and Character Recognition, pp.41-44, 1998.

[14] Y.-S. Jehng, L.-G. Chen and T.-D. Chiueh, "A Motion Estimator for Low Bit-rate Video Codec", IEEE Transactions on Consumer Electronics, Vol. 38, No. 2, pp. 671-674, 1992.

[15] Y.-S. Jehng, L.-G. Chen and T.-D. Chiueh, "An Efficient and Simple VLSI Tree Architecture for Motion Estimation Algorithms", IEEE Transactions on Signal Processing, Vol. 41. No. 2, pp. 889-900, 1993.

[16] M.-H. Sheu, J.-Y. Lee, J.-F. Wang, L.-W. Lee, S.-R. Mau and L.-Y. Liu, "An Architecture with Low Memory Bandwidth and Less Hardware Cost for 3SBM Algorithm", IEEE Region Ten Conference, pp. 559-562,

1993.

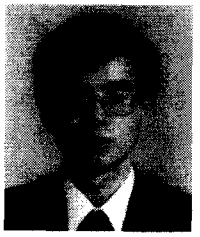
[17] H.-M. Jong, L.-G. Chen, and T.-D. Chiueh, "Parallel architectures of 3-step search block-matching algorithm for video coding", IEEE International Symposium on Circuits and Systems, pp. 209-212, 1994.

[18] H.-M. Jong, L.-G. Chen, and T.-D. Chiueh, "Parallel architectures for 3-step hierarchical search block-matching algorithm", IEEE Trans. on Circuits and Systems for Video Technology Vol. 4I, No. 4, pp. 407-416, 1994.

[19] A. Costa, A.D. Gloria, P. Faraboschi and F. Passaggio, "A VLSI Architecture for Hierarchical Motion Estimation", IEEE Transactions on Comsumer Electronics, Vol. 41, No. 2, pp. 248-257, 1995.

[20] S. Dutta and W. Wolf, "A Flexible Parallel Architecture Adapted to Block-Matching Motion-Estimation Algorithms", IEEE Trans. on Circuits and Systems for Video Technology, Vo. 6, No. 1, pp. 74-86, 1996.

[21] Y.-K. Lai, "A Memory Efficient Motion Estimator for Three Step Search Block-matching Algorithm", IEEE Trans. on Consumer Electronics, Vol. 47, No. 3, pp. 644-651, 2001.

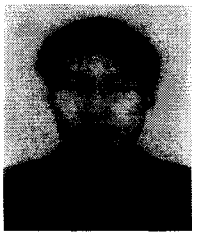


정 성 태

1987년 2월 서울대학교 컴퓨터공학과 졸업
1989년 2월 서울대학교 컴퓨터공학과 석사학위 취득
1994년 8월 서울대학교 컴퓨터공학과 박사학위 취득
1994년 9월~1995년 2월 한국전자통신연구소 박사후연수연구원
1999년 1월~1999년 12월 미국 Univ. of Utah 교환교수
1995년 3월~현재 원광대학교 전기전자및정보공학부 교수
관심분야 : 휴먼 컴퓨터 인터페이스, 영상 처리, VLSI/CAD

이 상 설

1984년 2월 고려대학교 전자공학과 졸업
1989년 2월 한국과학기술원 전기및전자공학과 석사학위 취득
1994년 2월 한국과학기술원 전기및전자공학과 박사학위 취득
1994년~현재 원광대학교 전기전자및정보공학부 교수
관심분야 : 병렬컴퓨터구조, SoC, 영상 및 통신 VLSI, 임베디드시스템



남 궁 문

1984년 2월 원광대학교 토목공학과 졸업
1986년 2월 전북대학교 토목공학과 석사학위 취득
1992년 3월 히로시마대학교 토목공학과 박사학위취득
1997년 11월~1998년 1월 미국 Univ. of Illinois 교환교수
1999년 12월~2000년 1월 Technische University Darmstadt 교환교수
2000년 7월~2000년 8월 Hiroshima University 교환교수
1992년 3월~현재 원광대학교 토목환경·도시공학부 교수
관심분야 : 교통행태, 인지공학, Simulation, CVM, GIS, Fuzzy, Neural Net