논문 2007-44SP-2-9

H.264 움직임 추정을 위한 효율적인 SAD 프로세서

(Efficient SAD Processor for Motion Estimation of H.264)

장 영 범*, 오 세 만**, 김 비 철**, 유 현 중*

(Young-Beom Jang, Se-Man Oh, Bee-Chul Kim, and Hyeon-Joong Yoo)

요 약

이 논문에서는 H.264의 효율적인 움직임 추정을 위한 새로운 SAD(Sum of Absolute Differences) 프로세서의 구조를 제안하였다. SAD 프로세서는 전영역 탐색기법의 움직임 추정이나 고속 탐색기법의 움직임 추정에서 모두 사용되는 중요한 블록이다. 제안된 구조는 SAD 계산기 블록, combinator 블록, 최소값 계산기 블록의 3개의 블록으로 구성된다. 제안된 구조는 덧셈 연산을 분산 연산(Distributed Arithmetic)을 사용하여 계산함으로써 구조를 단순화시켰다. 제안 구조를 HDL(Hardware Description Language)을 사용하여 실험한 결과 기존의 구조와 비교하여 39%의 게이트 카운트 감소효과를 보였다. 또한 FPGA를 사용하여 검증한 결과도 32%의 게이트 카운트 감소효과를 보였다. 따라서 제안된 움직임 추정용 SAD 프로세서는 칩의 면적이 중요한 변수인 H.264 칩에서 널리 사용될 수 있는 구조가 될 것이다.

Abstract

In this paper, an efficient SAD(Sum of Absolute Differences) processor structure for motion estimation of H.264 is proposed. SAD processors are commonly used both in full search methods for motion estimation and in fast search methods for motion estimation. Proposed structure consists of SAD calculator block, combinator block, and minimum value calculator block. Especially, proposed structure is simplified by using Distributed Arithmetic for addition operation. The Verilog-HDL(Hard Description Language) coding and FPGA(Field Programmable Gate Array) implementation results for the proposed structure show 39% and 32% gate count reduction in comparison with those of the conventional structure, respectively. Due to its efficient processing scheme, the proposed SAD processor structure can be widely used in size dominant H.264 chip.

Keywords: SAD(Sum of Absolute Differences), Motion Estimation, H.264, Distributed Arithmetic

I. 서 론

디지털 동영상 정보는 정보량이 방대하므로 코딩 방식에 대한 많은 연구가 진행되어왔으며, 특히 통신채널의 급속한 보급에 따라 압축률이 더욱 향상된 코딩 방식이 필요하게 되었다. 이와 같은 요구에 따라 H.264 표준이 탄생하였다[1]. H.264 표준 코딩 방식은 기존의 압축 방식보다 높은 압축률을 제공하므로 동일 전송 대역폭에서 더욱 개선된 화질을 전송할 수 있게 된다. 이와 같이 높은 압축률을 갖는 이유 중 하나가 H.264의 움직임 추정(ME, Motion Estimation)이 가변 블록크기를 갖고 있기 때문이다. 즉, H.264 코딩은 가변 블록크기의 사용을 통해 높은 압축률을 달성할 수 있으며, 화질의 개선도 이룰 수 있다. 그러나 이 같은 가변 블록크기의 움직임 추정은 많은 양의 계산을 필요로 한다. 움직임 추정 방식은 전 영역 탐색(Full Search) 방식과 고속 탐색 방식이 있다. 지금까지 여러 가지 방식의 전 영역 탐색 방식이 제안되었는데, 이들은 복잡한 구조를 갖고 있거나 복잡한 제어를 필요로 한다.[2][3][4][5][10] 고속 탐색 방식으로는 다이아몬드 탐색 기법[6]과 그 응용 기법들이 사용되고 있다.[7][8][9]

H.264 움직임 추정을 위한 전영역 탐색 방식이나 고속 탐색 방식 모두 기본적으로 16개의 4x4 SAD(Sum of Absolute Differences) 계산을 필요로 한다. 즉, 4x4 SAD 계산기를 한개 만들어서 16번을 반복하여 사용하거나

* 정회원, ** 학생회원, 상명대학교 정보통신공학과
(College of Engineering, Sangmyung University)
※ 본 논문은 한국 산업기술평가원의 신기술실용화기술개발 사업으로 수행한 연구결과입니다.
접수일자: 2006년8월7일, 수정완료일: 2007년2월21일

4x4 SAD 계산기를 16개 만들어서 한 번에 계산을 마칠 수도 있다. 또한 그 중간인 4개의 4x4 SAD 계산기를 만들어서 4번 반복하여 사용할 수도 있다. 이 논문에서는 4x4 SAD 계산기를 16개 사용하여 고속으로 SAD를 계산하는 방식을 제안한다. 16개의 4x4 SAD 계산기를 사용하므로 각각의 4x4 SAD 계산기의 하드웨어 구조를 작게 하는 것이 이 논문의 핵심이다.

Ⅱ. 제안된 SAD 프로세서 구조

2. 1 전체 구조의 블록도

H.264에서의 움직임 추정은 일반적인 고정 16x16블록으로 사용하는 것과 달리 7가지 종류의 가변블록 크기를 갖고 있다. 즉, 4x4 블록의 조합으로 다음 그림 1과 같이 16x16 크기의 블록을 구성한다.

H.264의 움직임 추정용 SAD 프로세서에서 많은 연산을 차지하는 것은 7가지의 가변블록의 SAD를 구하는 것이다. 그림 1에서 보듯이 16개의 4x4 블록들의 SAD를 구하면 나머지 다른 블록들은 이들을 조합함으로써 SAD를 구할 수 있다. 즉, 16개의 조합으로 총 41개의 SAD 값을 구할 수 있다. 16x16 SAD 연산은 크게 3 블록으로 구성된다. 첫 번째 블록은 4x4 SAD 계산기로서 4x4 픽셀의 SAD를 다음의 식(1)로 계산한다.

$$SAD(dx,dy)=\sum_{m=x}^{x+N-1}\sum_{n=y}^{y+N-1}|F_k(m,n)-F_{k-1}(m+dx,n+dy)|. \quad (1)$$

두 번째 Combination 블록에서는 16개의 4x4 SAD 계산 결과를 조합하여 41가지 종류의 SAD를 만들어낸다. 세 번째 최소값 계산기 블록에서는 다음의 식을 사용하여 최소의 SAD값을 갖는 움직임 벡터를 찾아내는 동작을 수행한다.

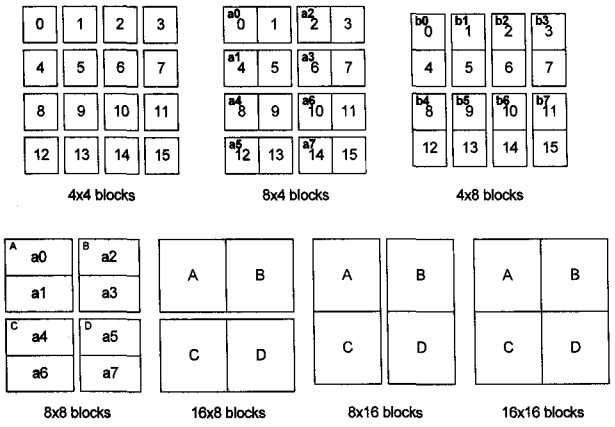


그림 1. 4x4 블록들로 구성된 H.264의 가변 크기의 블록

Fig, 1. H.264 variable blocks composed with 4x4 blocks.

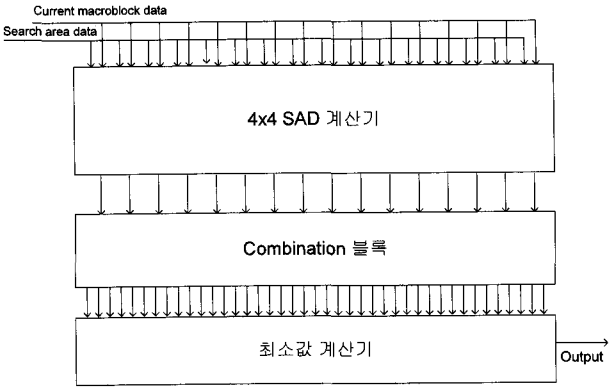


그림 2. 제안된 16x16 SAD 프로세서 구조

Fig. 2. Proposed 16x16 SAD processor structure.

$$\overrightarrow{MV}=(MV_x,MV_y)=\min_{(dx,dy)\in R^2}SAD(dx,dy).$$

우리가 제안하는 핵심 하드웨어는 첫 번째 블록인 4x4 SAD 계산기의 저전력 구조이다. 즉, 4x4 SAD 계산을 위한 저전력 구조를 제안하며, 이를 16개 사용하여 4x4 SAD 계산기 구조를 설계하였다. 제안된 전체의 SAD 프로세서 구조는 다음과 같이 3개의 파트로 구성된다.

그림의 첫 번째 파트인 4x4 SAD 계산기에서는 16개의 4x4 SAD를 계산한다. 즉, 16x16 마크로 블록은 16개의 4x4 블록으로 구성되므로 동시에 16개의 4x4 SAD를 계산하는 파트이다. 두 번째 파트인 Combination 블록에서는 4x4 SAD들을 조합하여 더 큰 크기의 SAD들을 계산한다. 즉, H.264에서 필요한 41가지의 SAD들을 만들어낸다. 마지막으로 세 번째 파트인 최소값 계산기에서는 각각의 블록크기에 대한 최소 SAD를 비교하여 탐색영역 내의 가장 작은 값을 골라내는 작업을 수행한다.

2. 2 제안된 4x4 SAD 계산기 구조

4x4 SAD 계산기는 절대 값을 계산하는 AD 연산기와 SAD를 계산하는 하드웨어로 구성된다. 먼저 현재 프레임과 참조 프레임의 탐색영역내의 비교블록의 각 픽셀들의 뺄셈의 절대 값인 AD(absolute deference)를 그림 3과 같이 계산하여야 한다.

그림 3의 A1 가산기는 전가산기 7개와 반가산기 1개가 사용된다. 그 결과 값의 MSB를 검사하여 0이면, 즉

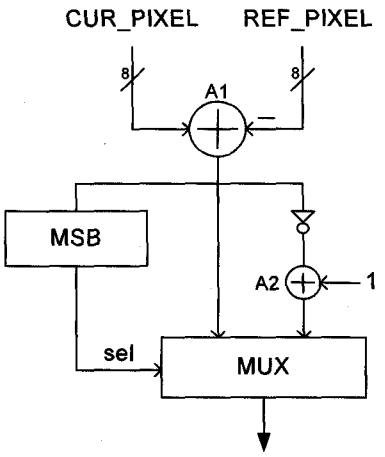


그림 3. 제안된 AD연산기
Fig. 3. AD calculator.

표 1. 예제의 16개의 AD값
Table 1. Example of 16 AD values.

AD0	001110101
AD1	000111100
AD2	010101010
AD3	011100011
AD4	001100111
AD5	010011100
AD6	010101010
AD7	000011101
AD8	001101010
AD9	000001101
AD10	000111110
AD11	001110101
AD12	010101011
AD13	000011100
AD14	000011110
AD15	010101001

양수이면 MUX의 sel이 0이 되어 값을 그대로 통과시키게 설계하였다. 그리고 MSB가 1이면, 즉 결과 값이 음수이면 절대값을 계산하기위하여 인버터회로와 +1회로를 거치도록 설계하였다. 이 동작이 직렬로 동작하므로 A2의 +1회로는 반가산기 1개로 구성할 수 있다.

이와 같이 직렬로 처리해도 되는 이유는 우리가 제안하는 다음단의 SAD 계산기가 bit-by-bit로 1 비트씩 처리하기 때문이다. 기존의 AD 연산기는 A1용으로 전가산기 7개와 반가산기 1개가 사용되며, 절대값 계산용으로 전가산기 8개와 반가산기 1개가 사용된다. 이와 같이 16개의 AD가 계산되면 그들의 합인 SAD를 구하여야 한다. 기존의 SAD 방식은 1단계에 8개의 가산기를 필요로 한다. 즉, AD0과 AD1, AD2와 AD3, AD4와 AD5, AD6과 AD7, AD8과 AD9, AD10과 AD11, AD12와 AD13, AD14와 AD15를 각각 가산기를 사용하여 더

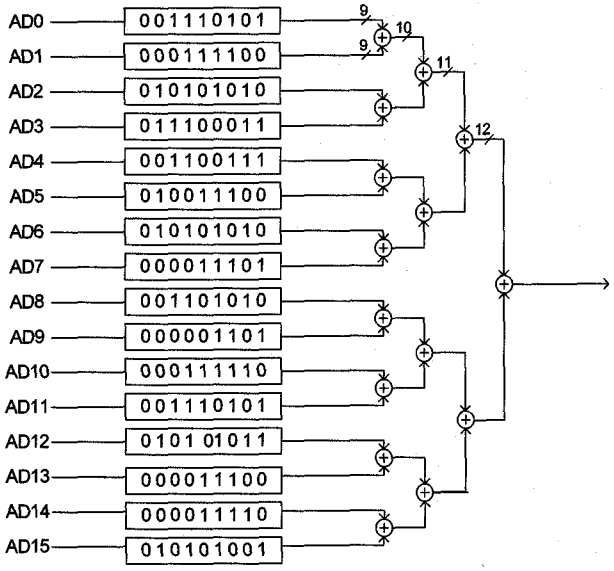


그림 4. 기존 방식의 4x4 SAD 계산기
Fig. 4. Conventional 4x4 SAD calculator.

한다. 따라서 2단계에는 4개, 3단계에는 2개 마지막 4단계에 1개의 가산기를 필요로 한다. 그림으로 아이디어를 쉽게 설명하기 위하여, 더해야 할 16개의 9비트 정세도의 AD 값들이 다음 표 1과 같다고 가정한다.

16개의 합을 구하기 위한 기존의 방식은 다음 그림 3과 같이 9비트부터 12비트의 15개의 가산기를 사용한다. 그림 4의 가산기에서, 전가산기와 반가산기의 게이트 수준으로 보면 전가산기가 131개, 반가산기 15개 사용된다.

기존의 SAD 방식 계산기 구조와 비교하여 우리는 16개의 operand를 LSB부터 더하는 방식을 제안한다. 즉, 16개의 AD 값들을 병렬로 받아서 LSB의 1비트부터 계산하는 방식이다.

이와 같은 연산을 분산 연산(Distributed Arithmetic) 방식이라 부르기도 한다. 4x4 SAD 출력 y는 다음과 같이 나타낼 수 있다.

$$y = \sum_{k=0}^{15} x_k \tag{3}$$

식 (3)에서 x_k는 16개의 AD 값들로서 다음과 같이 8비트로 나타낼 수 있다.

$$x_k = \sum_{n=1}^{8} b_{kn} 2^{-n} \tag{4}$$

식 (3)과 (4)를 결합하면 4x4 SAD 출력 y는 다음과 같이 나타낼 수 있다.

$$y = \sum_{k=0}^{15}[\sum_{n=1}^{8} b_{kn}2^{-n}] = \sum_{n=1}^{8}[\sum_{k=0}^{15} b_{kn}]2^{-n} \quad (5)$$

식 (5)를 계산하기 위한 제안된 구조는 그림 5와 같다. 그림 5에서 보면 1단계에서는 반가산기가 8개 필요하다. 2단계에서는 반가산기 4개와 전가산기 4개가 필요하다. 3단계에서는 3비트의 데이터를 더해야하므로 반가산기 2개와 전가산기 4개가 필요하다. 마지막으로 4단계에서는 4비트의 데이터를 더하므로 반가산기 1개와 전가산기 3개가 요구된다. 이렇게 4단계를 거치면 5비트의 LSB의 합이 출력된다. 이 결과 값은 1 비트 우로 쉬프트 되며 다음 클록에 (LSB-1)비트의 합과 더해진다. 이렇게 MSB까지 8 비트의 합이 모두 더해지면 SAD가 구해지는 구조이다.

그림 5의 전체 가산기를 게이트수준으로 보면 전가산기 26개 반가산기 16개를 사용하여 구현하였다. 기존의 그림 4 구조와 제안된 그림 5의 구조에 대한 가산기 수를 비교하면 다음 표와 같다.

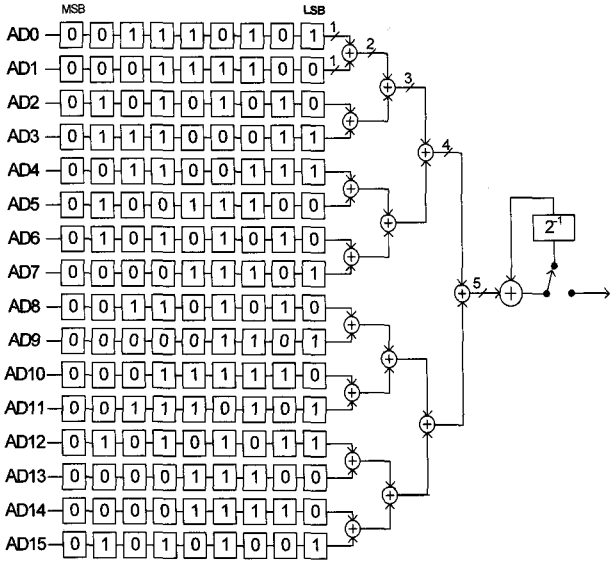


그림 5. 제안된 분산 연산 방식의 4x4 SAD 계산기
Fig. 5. Proposed 4x4 SAD calculator using Distributed Arithmetic.

표 2. 제안 구조의 전가산기와 반가산기의 수 비교
Table 2. Number of full adders and half adders in the proposed structure.

	기존 구조		제안 구조	
	전가산기	반가산기	전가산기	반가산기
AD 연산	240	32	112	32
SAD 연산	131	15	26	16
계	371	47	138	48
상대면적	1113	47	414	48
총상대면적	1160		462	
%	100		39.8	

표 2 에서 보듯이 기존 구조와 비교하여 전가산기의 수가 371개에서 138개로 감소되었으며, 반가산기의 수는 47개에서 48개로 1개가 증가하였다. 위의 표에서 상대면적의 계산은 반가산기 1개의 면적을 1로 정의하고 전가산기의 상대면적은 반가산기의 3배로 정의하였다. 그 결과 총상대면적은 기존구조가 1160이고 제안구조가 462로 예상된다. 따라서 제안 구조의 4x4 SAD 연산기와 기존 구조를 비교하였을 때 60.2%의 면적 감소가 예상된다.

2. 3 Combination block 구조

4x4 SAD 계산기의 결과 값들은 다음의 구조를 통하여 여러 가지 조합의 SAD를 계산할 수 있다. 즉, 41가지의 SAD 조합들을 만들어낸다. 그림 6에서 보듯이 Combinator의 출력은 총 41개이며, 41개의 레지스터에 그 값들이 저장된다. 4x4 SAD 값들은 직접 레지스터 0부터 레지스터 15까지의 16개의 레지스터에 저장된다. 8x4 SAD 값들은 레지스터 a0부터 레지스터 a7까지의 8개의 레지스터에 저장되며, 4x8 SAD 값들은 레지스터 b0부터 레지스터 b7까지의 8개의 레지스터에 저장된다. 8x8 SAD 값들은 레지스터 A부터 레지스터 D까지의 4개의 레지스터에 저장된다. 16x8 SAD 값들은 레지스터 AB와 레지스터 CD의 2개의 레지스터에 저장된다. 8x16 SAD 값들은 레지스터 AC와 레지스터 BD의 2개의 레지스터에 저장된다. 마지막으로 16x16 SAD 값은 직접 레지스터 ABCD에 저장된다.

2. 4 최소값 계산기

최소값 계산기 하드웨어는 41개로 구성되며, 1번의 블록 정합마다 새로운 값들이 최소값 계산기로 들어오게 설계하였다. 전 영역 탐색 알고리즘을 사용하는 경우에는 H.264의 탐색영역이 16x16이므로 256번의 블록 정합이 요구된다. 따라서 256개의 새로운 값들이 최소값 계산기로 입력되고, 그 중에서 가장 작은 값이 선택되도록 하기 위하여 다음과 같이 설계하였다.

제안한 구조는 그림 7의 최소값 계산기를 41개 사용한다. 그림 7에서 Reg.C에는 256번의 블록 정합마다 새 값들이 저장된다. 이 값에서 Reg.M의 값을 빼서 음수가 되면 새 값인 Reg.C가 더 작음을 의미한다. 이때에 1 bit register인 Reg.B가 1이 되어 AND 게이트가 열리

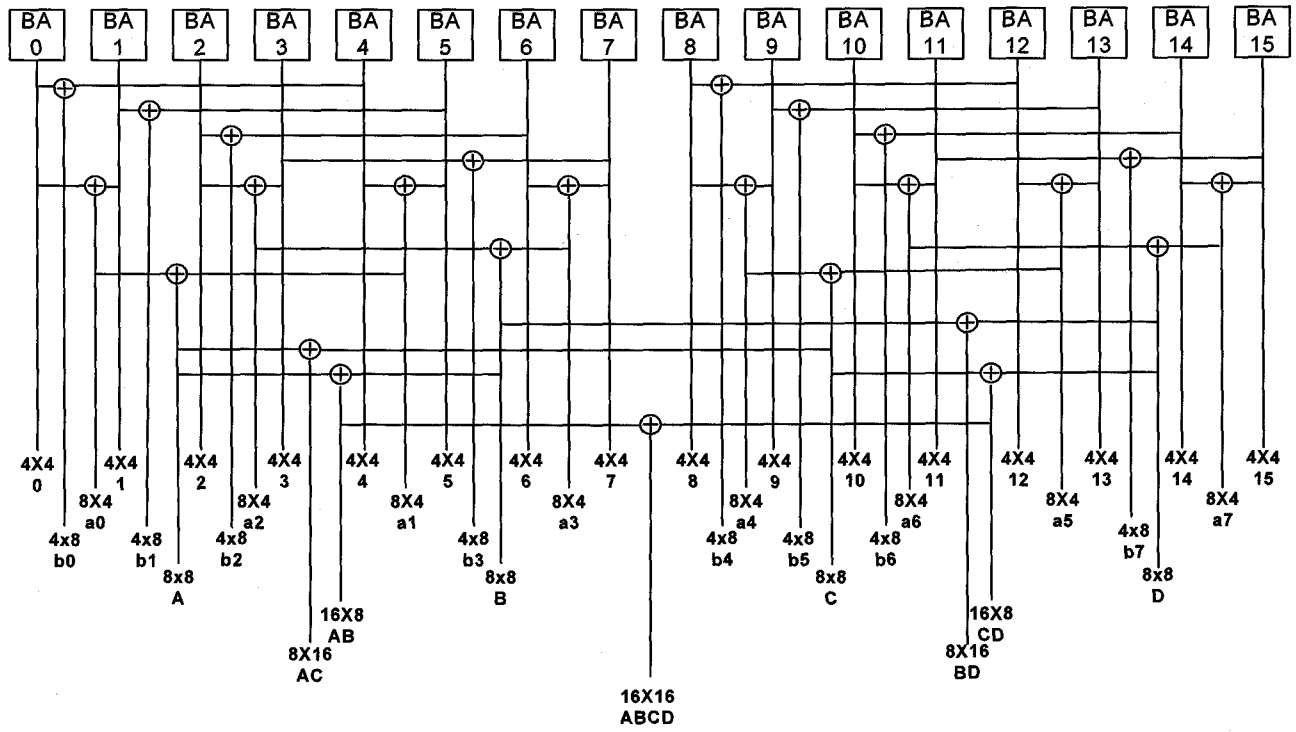


그림 6. 제안된 Combination 블록 구조
Fig. 6. Proposed Combination block structure.

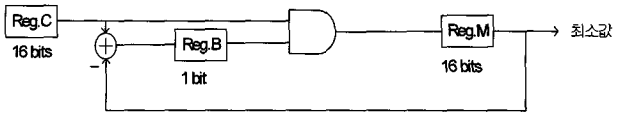


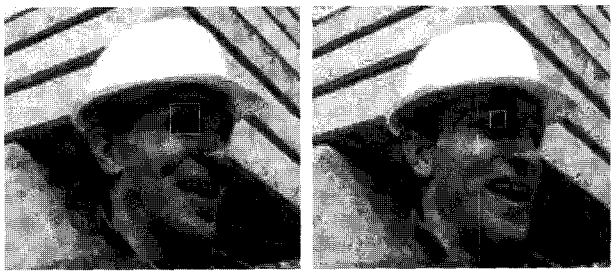
그림 7. 제안된 최소값 계산기 구조
Fig. 7. proposed minimum calculator stucture.

므로 Reg.M이 Reg.C의 값으로 바뀌게 설계하였다. 따라서 256번의 블록 정합이 끝나면 각각 41개의 Reg.M에 최소값이 선택되어 저장되도록 설계하였다. 이때에 최소값과 그 위치도 함께 저장된다. 우리가 제안한 알고리즘은 1번의 블록정합을 위하여 SAD 연산기에서 9클록이 필요하다. 따라서 총 256번의 블록 정합인 경우에는 2304 클록이 필요하게 된다.

Ⅲ. 실험 및 고찰

3.1 Verilog-HDL 시뮬레이션

본 논문에서 제안한 4x4 SAD 프로세서의 성능을 평가하기 위해 Verilog-HDL시뮬레이션을 수행하였다. 실험에 사용된 Tool은 Xilinx ISE 7.1i이고, 입력샘플로 그림 8과 같은 순차적인 2프레임의 QCIF급 "foreman" 영상을 사용하였다. 움직임 벡터의 범위는 (-16, -16)~(15, 15)를 사용하였다.



(a) (b)

그림 8. (a)foreman 참조영상, (b)foreman 현재영상
Fig. 8. (a)foreman reference data, (b)foreman current data.

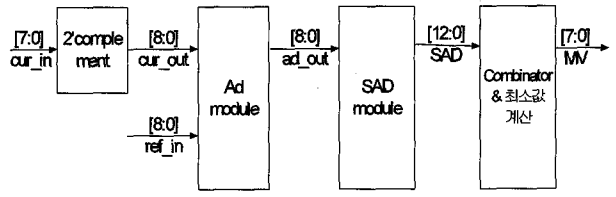


그림 9. SAD 프로세서의 시뮬레이션 블록도
Fig. 9. Simulation block diagram for proposed SAD processor.

전체 시뮬레이션의 블록도는 그림 9와 같다.

8 비트 정세도의 현재 영상의 각 픽셀 값은 AD연산기의 뺄셈 연산을 위해 9 비트 정세도의 2의 보수형 값으로 변환된다. 즉, 9 비트의 현재 프레임의 데이터

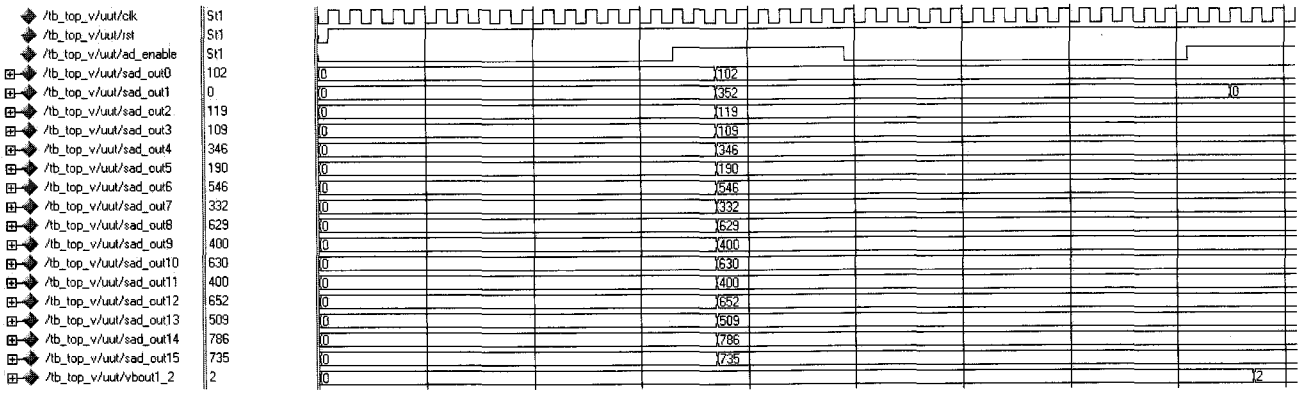


그림 10. 제안구조의 Verilog-HDL 시뮬레이션 결과
Fig. 10. Verilog-HDL simulation result for proposed structure.

값과 메모리에 저장되어 있는 9 비트의 참조 프레임 데이터는 AD 모듈의 입력으로 사용되고 9 비트의 AD값이 출력된다. 출력된 값은 다시 입력이 16개인 4x4 SAD 모듈의 입력으로 사용하였다. 기존 구조에 대한 실험에서는 처음 클록에 비트의 가산기를 8개 사용하고 8개의 10 비트 레지스터에 저장된다. 두 번째 클록에 10 비트의 가산기를 4개 사용하고 4개의 11 비트 레지스터에 저장된다. 세 번째 클록에 11 비트의 가산기를 2개 사용하고 2개의 12 비트 레지스터에 저장된다. 네 번째 클록에 12 비트의 가산기를 1개 사용하고 1개의 13 비트 레지스터에 저장되어 4x4 블록의 SAD 값이 출력되도록 실험하였다.

제안된 SAD 프로세서 구조에 대한 실험은 다음과 같이 수행하였다. 제안구조는 처음 클록에 1 비트의 가산기가 8개 사용되고 8개의 2 비트 레지스터에 저장하였다. 두 번째 클록에는 2 비트의 가산기를 4개 사용하고 4개의 3 비트 레지스터에 저장하였다. 세 번째 클록에 3 비트의 가산기를 2개 사용되고 2개의 4 비트 레지스터에 결과 값을 저장하였다. 네 번째 클록에 4 비트의 가산기를 1개 사용하고 1개의 5 비트 레지스터에 결과를 저장하였다. 다섯 번째 클록에 먼저 저장되어있던 값을 LSB 쪽(right shift)으로 1 비트 쉬프트 하고 현재 출력된 5 비트의 값과 덧셈을 수행하였다. 이렇게 비트를 순차적으로 연산하게 되어 총 13 클록 후에 SAD 값이 출력된다. 1개의 SAD 계산기는 16개의 AD 값과 1개의 SAD 값을 만들도록 실험하였다. 따라서 16x16의 마크로 블록에 대하여 제안된 구조는 16개의 SAD 값들을 동시에 출력하도록 실험하였다. 16개의 13 비트 정세도의 SAD 값들을 Combination 블록의 입력으로 사용하였다. 즉, Combination 블록으로 입력된 16개의 4x4 SAD 값들을 조합하여 4x8, 8x4, 8x8, 16x8, 8x16, 16x16 등의 크기의 총 41개의 SAD 값을 만들어 냈다. 41개의 Combination 블록 출력 값은 다음 새로운 값이 입력 될 때까지 레지스터에 저장하였다. 이와 같이 계산된 새로운 41개의 Combination 블록 출력 값이 최소값 계산기 블록으로 입력되면 기존에 저장되어 있던 값과 비교하여 더 작은 값을 다시 저장하도록 하였다. 이와 같은 연산을 반복하여 가장 작은 출력 값의 위치벡터인 8 비트의 움직임 벡터 값을 출력하게 된다. Combination 블록과 최소값 계산기는 기존구조와 제안구조에서 공동으로 사용하였다. 이와 같은 제안된 1개의 SAD 계산기 구조에 대한 Verlog-HDL 시뮬레이션을 수행하여 기존구조와 제안구조의 게이트 카운트를 구한 결과는 표 3과 같다.

표 3. 제안된 SAD 계산기의 게이트 카운트
Table 3. Gate count for proposed SAD calculator.

	AD	SAD	Total
기존구조	6,200	1,811	8,011
제안구조	3,914	998	4,912
제안구조의 %	63%	55%	61%

표 3에서 보듯이 기존구조의 게이트 카운트는 8,011이며, 제안구조는 4,912로서 39%의 게이트 카운트 감소를 달성하였다.

3. 2 16x16 SAD 프로세서의 FPGA 구현 결과

그림 10은 3. 1절에서 설계한 각각의 블록을 FPGA 구현을 통하여 움직임 벡터 값과 256개의 블록매칭 중에서 그림1 의 4X4 SAD 16개의 첫 번째 출력 값의 Timing diagram 결과화면이다. 각각의 4X4 SAD 계산기, Combination 블록, 최소값 계산기에 대한 FPGA 구현의 비교는 다음 표 4와 같다.

표 4. 제안된 16x16 SAD 프로세서의 FPGA 게이트 카운트

Table 4. FPGA gate count for proposed 16x16 SAD processor.

	기존구조	제안구조	제안구조의 %
SAD계산기	128,176	78,592	61%
Combination 블록	13,723	13,723	100%
최소값계산기	16,191	16,191	100%
Total	158,090	108,506	68%

제안 구조의 SAD 계산기는 기존구조에 비해 39%의 면적감소 효과를 얻을 수 있었고, 전체 SAD 프로세서 구조는 기조 구조와 비교하여 32%의 게이트 카운트 감소 효과를 달성하였다.

IV. 결 론

본 논문에서는 H.264 움직임 추정 알고리즘에 사용되는 SAD 연산을 위한 단순화된 구조를 제안하였다. 제안된 SAD 프로세서는 SAD 계산기, Combinator, 최소값 계산기로 구성하였으며 특히 분산 연산을 사용하여 SAD 계산기의 게이트 카운트를 줄일 수 있었다. 제안된 SAD 프로세서 구조에 대한 FPGA 실험 결과 게이트 카운트를 32% 감소시킬 수 있었다. 따라서 제안된 움직임 추정용 SAD 프로세서는 구현 면적을 중요한 변수로 설계하는 경우의 H.264 칩에서 널리 사용될 수 있을 것이다.

참 고 문 헌

[1] *Draft ITU-T recommendation and final draft International Standard of Joint Video Specification* (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC) JVT-G050, Geneva, Switzerland, 23-27, May 2003.

[2] S. Y. Yap and J.V. McCanny, "A VLSI Architecture for Variable Block Size Video Motion Estimation", IEEE Trans. on Circuits and Systems II: Express Briefs, Vol. 51, issue 7, pp. 384-389, Jul. 2004.

[3] Y. W. Huang, T.C. Wang, B.Y. Hsieh, and L. G. Chen, "Hardware Architecture Design for Variable Block Size Motion Estimation in MPEG-4 AVC/JVT/ITU-T H.264", Proceedings of the 2003 International Symposium on Circuits and Systems, Vol. 2, pp. 796-799, May 2003.

[4] C. Y. Kao and Y. L. Lin, "An AMBA-Compliant Motion Estimator for H.264 Advanced Video Coding", Proceedings of 2004 International Conference on SOC Design, Vol.1, pp. 117-120, Oct. 2004.

[5] C. Y. Cho, S. Y. Huang, and J. S. Wang, "An Embedded Merging Scheme for H.264/AVC Motion Estimation", Proceedings of International Conference on Image Processing, Vol. 1, pp. 909-912, Sep. 2003.

[6] S. Zhu and K. Ma, "A new diamond search algorithm for fast block matching", IEEE Trans. on Image Proc., Vol. 9, No. 2, pp.287-290, Feb. 2000.

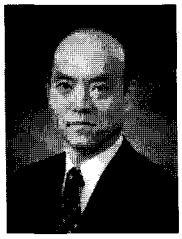
[7] P. Hosur and K. Ma, "Motion vector field adaptive fast motion estimation", Second International Conference on Information, Communication and Signal Processing (ICICS '99), Singapore, pp. 7-10, Dec. 1999.

[8] A. Tourapis, O. Au, and M. Liou, "Highly efficient predictive zonal algorithms for fast block-matching motion estimation", IEEE Trans. on Circuits and Systems for Video Technology, pp. 934-947, Oct. 2000.

[9] Donghyung Kim, Jechang Jeong, Xiao Song and Chengke Wu, "A Fast Macroblock Mode Selection Algorithm In The H.264/AVC Standard," IWAIT 2005, pp.157-162, Jan. 2005

[10] Chen-Fu Lin, Jin-Jang Leou, "Adaptive Fast Full Search Motion Estimation Algorithm for H.264," IEEE International Symposium on Circuits and Systems, pp.1493-1496, May 2005.

저 자 소 개



장 영 범(정회원)

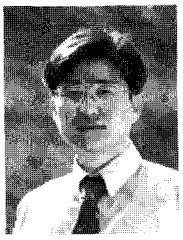
1981년 연세대학교 전기공학과 졸업(공학사)
1990년 Polytechnic University 대학원 졸업(공학석사)
1994년 Polytechnic University 대학원 졸업(공학박사)
1981년～1999년 삼성전자 System LSI 사업부 수석연구원
2000년～2002년 이화여자대학교 정보통신학과 연구교수
2002년～현재 상명대학교 정보통신공학과 교수
<주관심분야 : 통신신호처리, 비디오신호처리, SoC 설계>

김 비 철(학생회원)

2006년 상명대학교 정보통신 공학과 졸업 (공학사)
2006년～현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, SoC 설계>

오 세 만(학생회원)

2005년 상명대학교 정보통신 공학과 졸업(공학사)
2005년～현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, SoC 설계>



유 현 중(정회원)

1982년 서강대학교 전자공학과 졸업(공학사)
1991년 Missouri University 대학원 졸업(공학석사)
1996년 Missouri University 대학원 졸업(공학박사)
1982년～1989년 국방과학연구소
1992년～1995년 미국 앨러바마대학 방문학자
1996년 삼성전자
1999년～2001년 ㈜ACN Tech. ㈜알파텔레콤 연구소장
1996년～현재 상명대학교 정보통신공학과 교수
<주관심분야 : 인공신경망응용, 패턴인식, 영상/동영상 처리>