논문 2009-46SD-11-9

다양한 블록 크기의 전역 탐색 알고리즘을 위한 효율적인 구조를 갖는 움직임 추정기 설계

(The Motion Estimator Implementation with Efficient Structure for Full Search Algorithm of Variable Block Size)

황 종 희*, 최 윤 식*

(Jonghee Hwang and Yoonsik Choe)

요 약

움직임 추정은 영상 부호화 시스템에서 큰 비중을 차지하는 부분으로, 실시간 동작을 위해서는 효율적인 구조를 필요로 한다. 따라서 H.264 전체 시스템을 위한 움직임 추정기 블록의 구현은 부호화 과정을 고속으로 수행할 수 있도록 별도의 전용 하드웨어 모듈로 설계하는 것이 바람직하다. 본 논문에서는 많은 연산량을 효율적으로 줄일 수 있도록 병렬 처리를 바탕으로 움직임 추정 감지 블록, 41개의 SAD(Sum of Absolute Difference)값 계산 블록, 최소의 SAD값 계산과 움직임 벡터 생성 블록을 제안하고자 한다. 움직임 추정 감지 블록과 최소의 SAD값 계산기에서는 선계산(pre-computation) 방법을 적용함으로써, 입력 Switching Activity를 줄여 고속 구현이 가능하도록 하였으며, 움직임 추정 감지 블록과 41개의 SAD값 계산 블록에서 가장 많은 부분을 차지하는 가산기 구조를 일반적으로 사용되는 Ripple Carry Adder 대신에 Carry Skip Adder를 적용함으로써, Adder Tree 구조를 고속으로 처리할 수 있도록 하였다. 또한 외부에서 탐색 영역 제어와 같은 주요 변수를 쉽게 제어할 수 있도록 하여, 하드웨어 구조의 효율성을 높였다. 시뮬레이션 및 FPGA 검증 결과, 움직임 추정기의 임계 경로를 발생시키는 MED블록에서 일반적인 구조를 적용했을 때보다 19.89%의 Delay 감소 효과를 얻을 수 있었다.

Abstract

The motion estimation in video encoding system occupies the biggest part. So, we require the motion estimator with efficient structure for real-time operation. And for motion estimator's implementation, it is desired to design hardware module of an exclusive use that perform the encoding process at high speed. This paper proposes motion estimation detection block(MED), 41 SADs(Sum of Absolute Difference) calculation block, minimum SAD calculation and motion vector generation block based on parallel processing. The parallel processing can reduce effectively the amount of the operation. The minimum SAD calculation and MED block uses the pre-computation technique for reducing switching activity of the input signal. It results in high-speed operation. The MED and 41 SADs calculation blocks are composed of adder tree which causes the problem of critical path. So, the structure of adder tree has changed the most commonly used ripple carry adder(RCA) with carry skip adder(CSA). It enables adder tree to operate at high speed. In addition, as we enabled to easily control key variables such as control signal of search range from the outside, the efficiency of hardware structure increased. Simulation and FPGA verification results show that the delay of MED block generating the critical path at the motion estimator is reduced about 19.89% than the conventional strukcture.

Keywords: motion estimation detection, SAD, pre-computation, carry skip adder

Ⅰ. 서 론

디지털 멀티미디어 방송에 사용되는 H.264는 압축 효율이나 영상의 화질 측면에 있어서 기존의 표준보다

* 정회원, 연세대학교, 전기전자공학과
(Dept., Electrical & Electronics Eng. Yonsei University)
접수일자: 2009년9월1일, 수정완료일: 2009년10월23일

뛰어난 성능을 보이지만, 그에 따라 부호화 과정의 복잡도나 부호화 시간이 훨씬 증가하여 비디오 영상의 실시간 처리가 어렵다. 특히 움직임 예측 및 보상 과정은 영상 부호화 시스템에서 큰 비중을 차지하는 부분이며, 차세대 코덱으로 불리는 H.265에서도 움직임 추정(motion estimation)을 수행하는 부분은 전체 인코딩 시스템에서 가장 큰 연산량을 가질 것이다. 예를 들어 전역 탐색 알고리즘 (full search algorithm)은 가장 편리하면서도 가장 효과적으로 움직임 추정을 수행할 수 있는 방법이지만 다양한 블록 사이즈의 매크로 블록에 대한 움직임 벡터를 구하기 위해서는 허용된 탐색 영역내의 모든 픽셀에 대해 블록 단위로 SAD(sum of absolute difference) 연산을 수행해야 하기 때문에 매우 많은 연산량이 요구되어 진다. 실제로 전역 탐색을 통한 움직임 추정 알고리즘은 그림 1에서와 같이 일반적인 동영상의 부호화 시스템에서 전체 연산량의 65%이상을 차지할 정도로 큰 비중을 차지하고 있다.

일반적으로 IPBBPBBP … 구조의 비디오 영상에서 I 프레임을 제외한 P/B 프레임은 영상의 화질과 압축 효율이 움직임 추정에 의해 많은 영향을 받으며, H.264에서는 움직임 추정의 비용 함수로서 다음과 같은 RDO(Rate Distortion Optimization) 개념을 사용한다.

$$J(\mathbf{m},\lambda_{MOTION}) = SAD(s,c(\mathbf{m})) + \lambda_{MOTION} \cdot R(\mathbf{m}-\mathbf{p}) \quad (1)$$

여기서 $\mathbf{m} = (m_x, m_y)^T$은 움직임 벡터, $\mathbf{p} = (p_x, p_y)^T$는 예측된 움직임 벡터, λ_{MOTION}은 라그랑제 상수(Lagrange multiplier)를 나타낸다. s는 현재 블록, $c(\mathbf{m})$은 움직임 벡터 $\mathbf{m}$이 가리키는 참조 블록을 의미한다. H.264에서는 이와 같이 현재 블록과 참조 블록의 오차 또는 왜곡 뿐만 아니라 움직임 벡터의 부호화율(Bit Rate)을 함께 고려하여 최소 비용을 가지는 지점을 찾는 움직임 예측을 수행한다. 전역 탐색을 수행하면 탐색 영역 내의 모든 블록에 대해 위와 같은 비용함수 $J(\mathbf{m},\lambda_{MOTION})$를 계산하기 때문에 최소의 비용에 해당하는 블록을 찾을 수 있다. 여기서 움직임 추정을 위한 오차(Distortion) 측정 함수로 식(2)와 같이 Mean Square Error(MSE), Sum of Absolute Difference (SAD), Mean Absolute Difference(MAD)가 존재하며, 이 중에서 SAD가 곱셈기와 나눗셈 연산이 필요하지 않기 때문에 하드웨어 구현에 주로 사용된다.

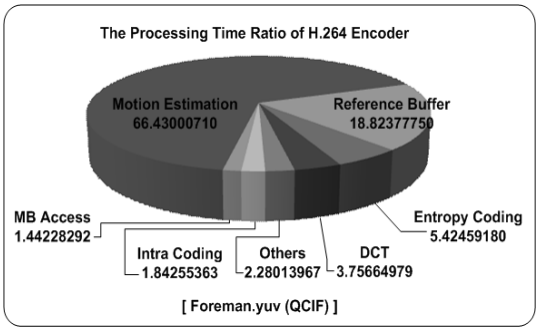

그림 1. H.264 부호화기의 각 모듈별 수행 시간 비율
Fig. 1. The processing time ratio of H.264 Encoder.

$$MSE = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}(f(i,j)-\hat{f}(i,j))^2$$
$$SAD = \sum_{i=1}^{N}\sum_{j=1}^{N}|f(i,j)-\hat{f}(i,j)| \quad (2)$$
$$MAD = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}|f(i,j)-\hat{f}(i,j)|$$

본 논문에서는 움직임 추정 알고리즘이 갖는 많은 연산량을 효율적으로 감소시키면서 고속 처리가 가능한 움직임 추정 블록을 설계하기 위해서, SAD를 계산하기 위한 픽셀 연산 블록들을 모두 병렬 연산이 가능하도록 하였다. 또한, 전체 움직임 추정기를 구성하는 움직임 추정 감지 블록과 41개의 SAD값 계산기 블록에서 가장 많은 부분을 차지하는 가산기 구조를 일반적으로 사용되는 Ripple Carry Adder 대신에 Carry Skip Adder를 적용함으로써, Adder Tree 구조를 고속으로 처리할 수 있도록 하였고, 움직임 추정 감지 블록과 최소의 SAD값 계산기에서는 선계산(pre-computation)을 이용함으로써 입력 Switching Activity를 줄여 고속 구현이 가능하도록 하였다.

논문의 구성은 다음과 같다. Section Ⅱ에서는 전역 탐색 알고리즘 및 기존 논문 연구에 대하여 정리하였고, Section Ⅲ에서는 다양한 블록 사이즈의 전역 탐색에 적합한 움직임 추정기 구조를 제안하였다. Section Ⅳ에서는 Verilog HDL(Hardware Description Language)를 사용하여 움직임 추정기를 하드웨어로 구현하고, FPGA(Field Programmable Gate Array)로 검증한 결과를 정리하였다. Section Ⅴ에서는 제안된 구조의 우수성을 설명하면서 결론을 맺는다.

Ⅱ. 전역 탐색 알고리즘 및 기존 논문 연구

1. 다양한 블록 크기의 전역 탐색

움직임 추정 방법은 Optical flow method,

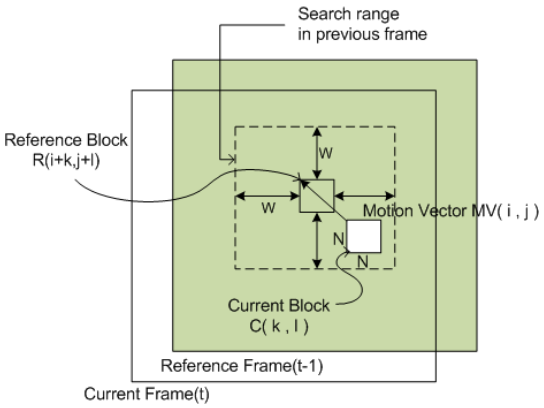

그림 2. 블록 정합을 통한 움직임 추정 방법
Fig. 2. The motion estimation method through block matching.

Pel-recursive method, Phase correlation method, Block matching method 등으로 구성되었으며, 이 중에서 간단한 동작과 하드웨어 구현으로 인해 Block matching method가 가장 널리 사용되고 있다[1]. 전역 탐색 block matching method는 그림 2와 같이 현재 프레임에서 N×N 크기의 현재 블록(current block)에 대해서 이전 프레임의 탐색영역에서 모든 참조 블록(reference blocks)과의 정합을 통하여 정합오류가 최소가 되는 탐색 블록을 찾아 이를 이용하여 움직임 벡터를 추정하는 방식이다. 이를 수식으로 표현하면 아래와 같다.

$$SAD(i,j) = \sum_{k=1}^{N}\sum_{l=1}^{N} |C(k,l) - R(i+k,j+l)|$$
$$Best\,Match = \min(SAD(i,j)) \quad (3)$$

C(k,l)과 R(i+k,j+l)은 현재 프레임과 참조 프레임의 픽셀위치에서 휘도값을 나타내고, Best Match는 탐색영역에서 최소의 SAD값을 갖는 움직임 벡터 MV(i,j)를 나타낸다.

블록 정합은 두 블록간의 유사성을 계산하는 과정이고, 프레임간 시간적 차이가 매우 작기 때문에 두 프레임은 많은 연관성을 갖게 된다. 연속된 두 프레임간 이러한 관계를 Temporal Redundancy라고 하며 영상 압축이란 중복성을 제거하는 것이기 때문에 움직임 추정은 Temporal Redundancy를 제거하는 과정이라고 할 수 있다. H.264에서는 가변 블록 움직임 보상을 사용하고 있다. MPEG-2에서는 16×16 화소 고정크기 움직임 보상 블록을 사용하고, MPEG-4에서는 16×16 화소와 8×8 화소 두 종류의 움직임 보상 블록을 사용한다. 반면에 H.264에서는 그림 3과 같이 16×16 화소로부터 4×4 화소까지 7 종류의 움직임 보상 블록크기를 사용하고 있으며, 움직임 추정기에서 발생하는 출력인 SAD 최소값과 움직임 벡터는 4×4 16개, 4×8 8개, 8×4 8개, 8×8 4개, 8×16 2개, 16×8 2개, 16×16 1개로 총 41개로 나타낼 수 있다. 매크로 블록 모드는 8×8, 8×16, 16×8, 16×16으로, 서브 매크로 블록 모드는 4×4, 4×8, 8×4, 8×8으로 구성된다. 움직임 보상의 블록크기를 보다 작게 하면 보다 정확한 예측을 할 수 있어, 압축 효율을 높일 수 있다.

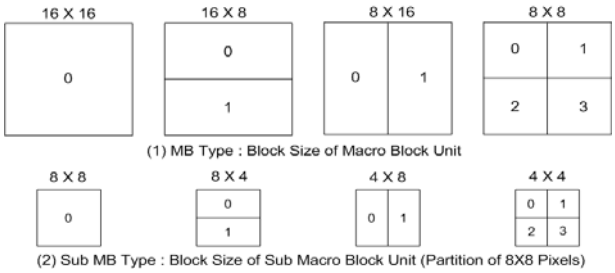

그림 3. H.264의 다양한 블록 사이즈
Fig. 3. The various block sizes in H.264.

2. 기존 논문 분석

움직임 추정을 위한 많은 알고리즘과 하드웨어 관련된 논문들이 제안되어왔다. 연산 복잡도를 줄이기 위해서 전역 탐색과는 달리 탐색 영역내에서 몇 개의 픽셀 위치에서만 탐색하거나 영상의 특성에 따라 블록을 선택하는 고속 알고리즘이 제안되었다[2~6]. 하지만, 이와 같은 고속 탐색 알고리즘은 움직임 예측의 시간이나 비용은 절약할 수 있을지 모르지만, 영상에 있어서 불규칙적인 메모리 제어, 화질의 열화, 압축율의 저하 등의 문제로 이어지게 된다. 따라서 전역 탐색 알고리즘이 구조의 간단함, 규칙성, 우수한 화질로 인해 실제 구현 과정에서 널리 사용되고 있다.

하드웨어 관점에서는 움직임 추정기를 구성하는 단위 블록들을 병렬로 배열하여 저전력 및 고속 처리가 가능한 구조가 제안되었고[7~8], 이전 프레임의 메모리 데이터 재활용을 극대화하기 위해서 탐색 영역내에서의 다양한 스캔 방법[9~10]과 대부분의 블록 정합이 탐색 영역 원점 주위에 분포되어 탐색되는 영역을 단순화 시키는 방법[11~12]들이 제안되어왔다. 이와 같은 방법은 최소의 SAD를 갖는 Best Matching 블록을 빠른 시간 내에 찾게 되면 연산량을 감소시키면서 고속처리가 가능하다. 그리고 메모리 측면에서는 탐색영역에서의 중첩

된 데이터를 재활용하여 메모리 대역폭을 줄일 수 있는 장점이 있다. 본 논문에서는 기존에 제안된 메모리 대역폭의 효율적인 제어와 병렬 처리를 기본 구조로 외부 인터페이스 회로 내장, 선계산 방법 적용, 움직임 추정기에서 대부분의 연산을 차지하는 가산기를 Carry Skip Adder로 변경 적용 등을 통해, 하드웨어 구조의 효율성을 향상시키면서 고속 처리가 가능하도록 하였다.

Ⅲ. 제안된 움직임 추정기 구조

1. 전체 움직임 추정기 구조의 블록도

움직임 추정기는 크게 현재 프레임과 참조 프레임의 영상 데이터를 저장하는 외부 메모리(SDRAM), 현재 프레임에서 16×16 매크로 블록 모드로 분할한 데이터와 최대 탐색영역의 데이터를 저장하는 내부 메모리(SRAM), 움직임 추정을 위한 하드웨어 로직 설계부, 탐색 영역의 크기에 따라 외부 메모리부터로 내부 메모리 데이터를 읽어 들이기 위한 주소 생성기 및 데이터 제어 설계부, 외부에서 사용자가 PC를 통해 주요 변수를 손쉽게 제어할 수 있도록 하기 위해 설계된 I2C(Inter Integrated Circuit) 통신용 인터페이스 회로부 등으로 구성된다. 여기서 SRAM은 FPGA에 내장된 형태로 구현하여 실시간으로 데이터 처리가 가능하도록 하였으며, ASIC(Application Specific Integrated Circuit)으로 제작시에도 SRAM은 내장되어 구현될 수 있다. 움직임 추정기에 대한 하드웨어 설계부는 16×16 픽셀을 4×4 픽셀 단위로 분할하여 현재 픽셀과 참조 픽셀간의 SAD를 계산하는 블록, 16개 4×4 픽셀의 SAD 값을 사용하여 7가지 모드인 4×4, 4×8, 8×4, 8×8, 8×16, 16×8, 16×16 픽셀에 대한 SAD값을 계산하는 블록, 모든 탐색 영역내에서 최소의 SAD값과 움직임 벡터를 계산하는 블록으로 구성된다. 위에서 설명된 전체 구조의 데이터 흐름을 그림 4와 같이 나타내었다. 움직임 감지를 위한 메모리 데이터 할당 방법, 움직임 추정기에 대한 하드웨어 설계부, I2C 통신을 위한 인터페이스 회로부에 대해서는 다음 절에서 상세하게 설명하였다.

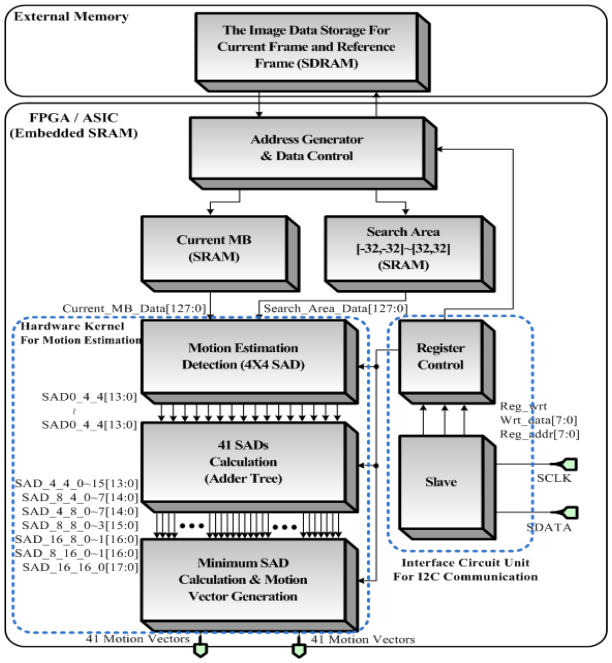

그림 4. 제안된 움직임 추정기의 전체 구조
Fig. 4. The whole architecture of proposed motion estimator.

2. 움직임 감지를 위한 메모리 데이터 할당 방법

탐색 영역내에서 전역 탐색을 수행할 때, 16×16 현재 블록은 SRAM에서 데이터를 받아온 후, 전역 탐색이 끝날 때까지 변경되지 않는다. 그러나 탐색 영역내에서 현재 블록이 한 픽셀씩 이동해 가며 Block Matching을 진행할 때, SRAM에서 16×16 픽셀에 해당하는 탐색 영역 데이터가 갱신되어야 한다. 여기서 중복되는 많은 픽셀 데이터를 효율적으로 재사용하기 위하여 메모리 재사용 방식과 그림 5의 화살표 방향과 같은 스캔 방법을 사용하였다. 그림 5에서와 같이 홀수 라인에서는 오른쪽으로 이동하고 짝수 라인에서는 왼쪽으로 이동하며 탐색영역 내에서 현재 매크로 블록과의 매칭을 진행한다. 그리고 각 라인의 마지막 탐색 지점의 연산을 완료하면 밑으로 이동하게 된다. 메모리 재사용 방식이란 입력받은 영상 데이터 중에서 16×16

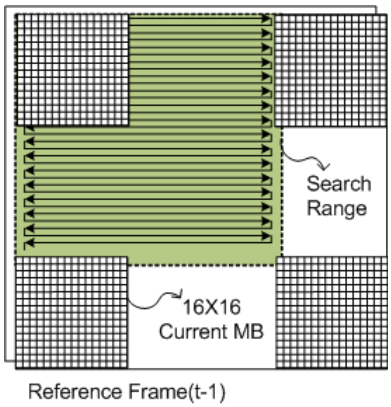

그림 5. 탐색 영역내에서 16×16 현재 블록의 스캔 방향
Fig. 5. The scan direction of a 16×16 current block in the search range.

픽셀 단위로 움직임 추정 후 다음에 입력받을 탐색 영역의 영상 데이터가 동일할 시 새롭게 영상 데이터를 입력받는 것이 아니라 메모리의 위치를 이동하여 영상 데이터의 입력 횟수를 줄이는 방식이다[12]. 위와 같은 방법을 적용하기 위해서, 그림 4에서 움직임 추정 감지 블록(MED)은 스캔 방향에 따라 좌·우 방향으로 이동할 때에는 탐색 영역내에서 16×1 픽셀에 해당하는 데이터만 입력받고, 연속해서 16×15 픽셀 데이터를 중복해서 사용할 수 있게 된다. 동일하게 아래 방향으로 이동할 때에는 1×16 픽셀에 해당하는 데이터만 입력받고, 15×16의 픽셀 데이터는 MED블록에서 이동되어 다음 연산 과정에서 재활용된다. 따라서 사용되는 SRAM은 실시간 처리가 가능하도록 16개로 나누어 클럭마다 탐색 영역내에서 16×16 매크로 블록과 정합될 새로운 데이터인 1×16 또는 16×1의 데이터를 읽어 들일 수 있도록 하였다. 즉, 각각의 SRAM은 하나의 픽셀에 대한 8비트 데이터 버스를 통해 출력하고, MED 블록에서는 16개의 SRAM에서 출력된 8비트 데이터를 묶어서 총 128비트를 가지고, SAD 연산을 위해 사용되는 단위 블록으로 재할당되게 된다. 또한 이 전 클럭에 유지하고 있던 픽셀 데이터는 SAD를 구하기 위하여 배열된 단위 블록내에서 스캔 방향과 데이터 입출력 포트를 통해 이동되어 재사용된다.

3. 움직임 추정 감지 블록(MED)

움직임 추정 감지 블록은 16×16의 현재 블록과 참조 블록 픽셀을 16개의 MED_X 블록에 할당하여 병렬 처리가 가능하도록 하였다. 이를 통해 로직이 차지하는 면적은 증가하지만, 처리 시간은 단축된다. MED_X 블록은 다양한 움직임 보상의 블록크기에서 가장 작은 4×4 SAD를 계산하는 역할을 수행하며, 병렬로 배열된 16개의 MED_X 결과가 41개의 SAD 계산 블록에 전달된다. 각각의 MED_X 블록은 그림 6과 같이 16개의 AD(Absolute Difference)블록과 Adder Tree를 갖는 32비트 입·출력 포트로 구성되어 있으며, 4×4 픽셀에 대한 SAD를 계산하여 출력하는 역할을 수행한다. 그리고 AD블록은 픽셀 단위의 절대값 연산을 수행하는 블록으로 8비트 입·출력 포트로 구성되며, AD_0, 4, 8, 12는 상위 블록인 MED_X블록을 통해 새롭게 SRAM에서 탐색영역의 데이터를 입력으로 받고 이전에 입력받은 데이터를 출력포트를 통해 전달하는 기능을 수행한다. 스캔 방향에 따라 SRAM으로부터 새롭게 입력받는

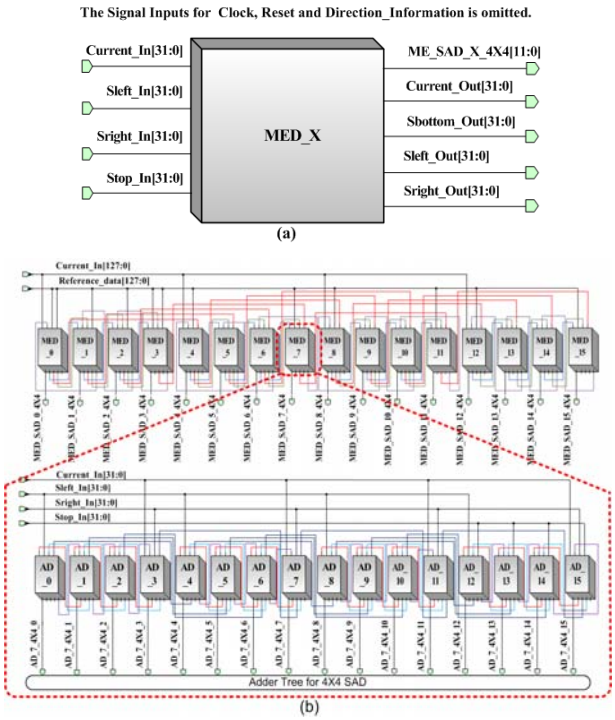

그림 6. (a) MED_X 블록의 입출력 포트, (b) MED 블록의 하드웨어 구조도

Fig. 6. (a) The input and output port of MED_X block, (b) Hardware structure of MED block.

AD블록과 기존의 데이터를 전달하는 방향은 움직임 추정기에서 입력되는 Select 신호에 따라 다르게 설정된다. 만약 스캔 방향이 오른쪽이라면, AD_0, 4, 8, 12 → AD_1, 5, 9, 13 → AD_2, 6, 10, 14 → AD_3,7,11,15 방향으로 참조 픽셀의 데이터가 전달된다. 그러나 SRAM에서 입력 받은 현재 블록의 데이터는 탐색 영역내에서 블록 매칭이 끝날 때까지 유지된다.

가. AD 블록과 4×4 SAD Adder Tree 구조

AD 블록은 현재 픽셀과 이전 픽셀과의 절대값을 구하는 블록으로, 입력으로 받는 두 개의 값을 비교하여 음의 값이 발생하지 않도록 뺄셈을 수행한다. 그 다음 구해진 16개의 차이값을 입력으로 Adder Tree구조를 통해 4×4 SAD값을 구하게 된다. 그림 7에서와 같이 다단으로 구성된 Adder Tree는 1단에서 8개, 2단에서 4개, 3단에서 2개, 4단에서 1개의 Ripple Carry Adder를 사용하여 구현된다. Ripple Carry Adder는 비트 단위 연산을 통해 그 다음 비트 연산에 대한 Carry를 발생하는 구조로 많은 처리 시간을 필요로 하며, 구조가 단순하여 널리 사용된다.

일반적으로, 움직임 추정기 구조에서 MED 및 AD

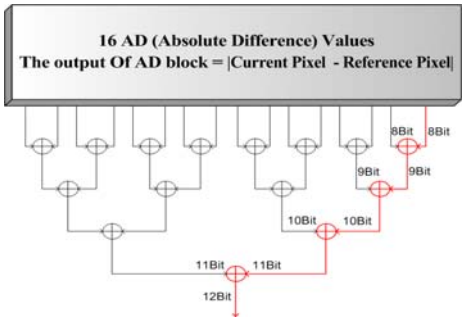

그림 7. 일반적인 4×4 SAD 계산기 구조
Fig. 7. The conventional structure of 4×4 SAD calculator.

블록들이 모두 병렬로 배열되어 처리될 때 AD블록과 Adder Tree의 데이터 처리 시간이 하드웨어 로직에 대한 임계 경로(Critical Path)가 된다. 따라서 전체 시스템의 처리 속도를 개선하기 위해서는 반드시 AD블록과 Adder Tree의 구조를 고속 처리가 가능하도록 최적화 시켜야 한다.

따라서 본 논문에서는 첫 번째로 절대값을 계산할 때 입력의 일부분만으로 출력을 나타낼 수 있는 입력들의 부분 집합을 찾아내서, 이를 이용하여 고속으로 출력될 수 있는 선계산(pre-computation)기법을 제안하였다. 즉, 그림 8과 같이 RB-CB와 CB-RB값 두 가지 경우에 대해 미리 뺄셈을 수행한 후, 현재 픽셀(CB[7])과 이전 픽셀(RB[7])의 최상위 비트값을 비교기의 입력으로 받아 미리 예측함으로써, 양이 되는 출력값을 선택할 수 있도록 하였다. 만약에 $CB^{[7]}$값과 $RB^{[7]}$값이 같다면 기존 구조와 동일하게 절대값을 구하게 된다. 위와 같이 제안된 선계산 구조는 입력의 Switching Activity를 줄여서 고속 구현을 가능하게 한다.

두 번째로 Adder Tree에 사용되는 가산기를 고속처리가 가능한 CSA(Carry Skip Adder) 구조로 변경 적용하였다. 가산기는 디지털 회로에서 가장 기본적이고

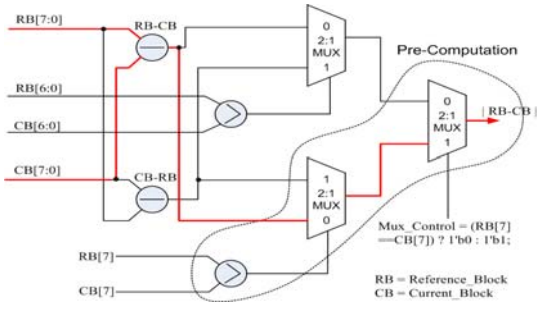

그림 8. 제안된 AD 구조
Fig. 8. The proposed AD Structure.

표 1. MED 블록에 사용된 가산기 종류 및 개수
Table 1. Adder type and number used for MED block.

가산기 종류	사용된 개수	비 고
8비트 가산기	128(8×16)	MED블록은 16개의 MED_X블록으로 구성.
9비트 가산기	64(4×16)	
10비트 가산기	32(2×16)	
11비트 가산기	16(1×16)	
Total	240	블록 정합당 사용된 가산기 개수

표 2. 8/16비트 가산기 유형에 대한 상대적인 비교.
Table 2. Relative comparison for 8/16Bit adder type.

Adder Type	Area(mm^2)		Delay(ns)		Avg. Power(uW)	
	8	16	8	16	8	16
Ripple Carry	1	1	1	1	1	1
Carry Skip	**1.23**	**1.45**	**0.82**	**0.60**	**1.15**	**1.06**
Carry Select	1.33	1.63	0.78	0.54	1.24	1.35
Lookahead	1.41	1.61	0.74	0.54	1.24	1.20

중요한 회로이며, 가산기 회로의 로직 처리 시간이 전체 디지털 회로에 많은 영향을 끼친다. 또한, 움직임 추정기의 대부분이 메모리 제어와 가산기로 구성되어 있음을 비추어 보았을 경우, 고속의 가산기 구조를 사용하는 것은 중요한 의미를 갖게 된다. 16×16 현재블록을 이전 탐색영역에서 한 번의 블록 정합을 수행할 경우 사용되는 가산기의 개수를 정리하면 표 1과 같이 나타낼 수 있다. MED 블록에서 16개의 4×4 SAD를 구하기 위하여 총 240개의 가산기가 필요함을 알 수 있다.

표 2는 0.8um 표준 셀 공정을 사용하여 각 가산기 종류에 따른 칩 면적, 게이트 지연, 평균 전력 소모량을 나타내었다[13~14]. RCA(Ripple Carry Adder)에 대한 결과값을 "1"로 정규화한 다음 나머지 가산기에 대해 결과값을 산출하였다. 본 논문에서는 표 2의 결과를 토대로, SAD를 구하기 위한 Adder Tree의 가산기로 상대적으로 적은 칩 면적의 증가로 고속 처리 및 전력 소모량 최소화가 가능한 CSA 구조를 사용하였다.

그림 9는 8비트 CSA 구조를 나타내며, 4비트 단위의 RCA에서 발생하는 Carry 신호를 예측하여 Carry 전파에 따른 시간 지연을 줄일 수 있도록 하였다. CSA는 4비트 RCA를 구성하는 4개의 전가산기에 비트단위로 데이터를 입력하고, 각 전가산기의 Carry 신호를 모두 And Gate연산을 통해 Carry를 Skip해야 할지 결정한다. 또한, 4비트 RCA가 연이어 사용되었을 경우에는 전단의 4비트 RCA의 Cout_1 Carry 신호와의 And

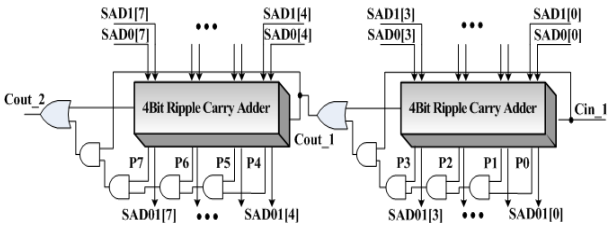

그림 9. 8비트 Carry Skip Adder 구조
Fig. 9. 8Bit structure for Carry Skip Adder.

Gate 연산을 통해 4비트 이상의 CSA 구조에 대한 Carry Skip을 결정하게 된다. 만약에 모든 Carry 신호(P7~P1, Cin_1, Cout_1)가 "1"이면, Cout_2는 Cin_1이 되어 Carry Skip이 된다. 따라서 표 2에서와 같이 CSA의 입력 비트수가 8비트에서 16비트로 증가하면 로직 Delay 시간이 8비트 입력보다 더 많이 감소하게 된다.

4. 41개의 SAD값 계산 블록

41개의 SAD값 계산(41 SADs Calculation) 블록은 MED 블록에서 생성된 16개의 4×4 SAD 값들을 사용하여 8×8, 8×16, 16×8, 16×16의 매크로 블록 모드와 4×4, 4×8, 8×4의 서브 매크로 블록 모드에 대한 SAD값을 계산한다. 4×4 SAD와 동일하게 Adder Tree 구조를 통해 구현이 되었으며, 3개의 4비트 RCA를 직렬로 연결한 CSA 구조를 적용하였다. 다단으로 구성된 Adder Tree를 통해서 1단에서는 12비트 가산기 16개를 사용하여 4×8, 8×4 SAD값을 구하고, 이 결과를 이용하여 2단에서는 13비트 가산기 4개를 사용하여 8×8 SAD값을 생성한다. 마찬가지로, 3단에서는 14비트 가산기 4개를 사용하여 8×16, 16×8 SAD값과 4단에서는 15비트 가산기 1개를 사용하여 16×16 SAD값을 각각 구한다.

5. 최소 SAD값 계산 및 움직임 벡터 생성 블록

최소값 계산기 블록은 그림 10 (a)의 하단부 점선 영역 41개가 병렬로 배열되어 있으며, 매 클럭마다 7가지의 매크로 및 서브 매크로 블록에 대한 새로운 SAD값 41개가 각 영역으로 입력된다. 최소값 생성 과정은 그림 10(a)와 같이 입력되는 SAD값을 Register에 저장하고, 동시에 이전 클럭에 최소의 SAD값이 저장되어 있는 Register값과 비교하여 입력되는 SAD값이 더 작으면 1비트 MUX에서 '1'을 출력하여 새로운 값으로 최소의 SAD에 대한 Register값을 갱신한다. 그렇지 않은 경우에는 MUX에서 '0'을 출력하여 기존의 최소 SAD값이 유지된다[15]. 탐색 영역이 (-16,-16)~(15,15)인 경우에는

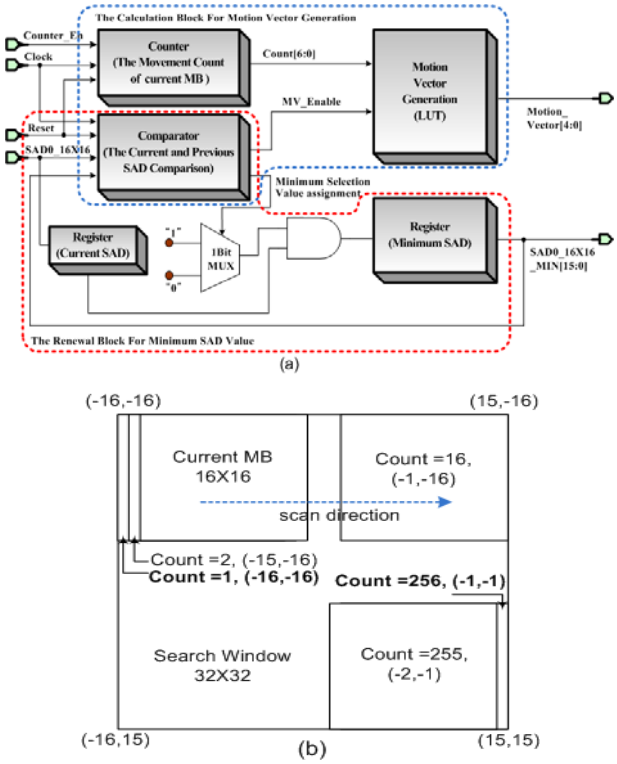

그림 10. (a) 최소 SAD 계산 및 움직임 벡터 생성 구조, (b) 움직임 벡터 생성 방법
Fig. 10. (a) Minimum SAD calculation and motion vector generation unit's architecture. (b) The generation method of motion vector.

256번의 블록 정합이 요구되므로, 매 클럭마다 41개의 값이 입력되어 256번의 최소값 계산을 수행하게 된다. 최소값 계산 블록에서는 그림 8의 제안된 AD 구조와 같이 최소값 계산기의 비교기에 대해 선계산 기법을 이용하여 구현함으로써, 입력 Switching Activity를 줄일 수 있도록 하였다. 다음은 움직임 벡터 생성 방법으로, 그림 10(b)와 같이 탐색 영역내에서 현재 블록이 이동하는 횟수를 Count함으로써, 움직임 벡터를 구할 수 있다. 현재 블록이 왼쪽 윗 부분부터 이동을 하기 때문에 이 부분의 Count가 1이 되고, 이동하면서 탐색 영역의 마지막 부분에 도달했을 때에는 Count가 256이 된다. 그리고 Count에 해당되는 움직임 벡터값은 LUT(Look-Up Table)형태로 저장하여 Count값에 맞게 출력될 수 있도록 하였다. 비교기 블록에서 매 클럭마다 입력되는 41개의 SAD값과 기존의 41개의 값과 비교하여, 더 작은 값이 입력되면 기존에 저장되어 있는 움직임 벡터값을 대신하기 위해서 Count값에 따라 움직임 벡터값을 저장하고 있는 LUT에 Enable 신호를 출력하여 움직임 벡터값을 바꾸게 된다. Counter값이 256를 가리키면, 다음 클럭에

Count 및 움직임 벡터값은 초기화된다.

6. I2C 통신을 위한 인터페이스 회로 블록

일반적으로, I2C(Inter Integrated Circuit)는 주변 장치를 단지 두 가닥의 신호선(직렬 데이터와 직렬 클럭)으로만 연결하여 동작하는 양방향 직렬 버스 규격을 말한다[16]. 여기서, I2C는 움직임 추정에 대한 하드웨어 블록을 FPGA 또는 ASIC으로 구현한 후 주요 신호를 PC가 Master가 되어 제어하기 위해 사용되었다. 그림 4의 Slave 블록은 외부에서 풀업 저항이 연결된 직렬 데이터(SDA)와 직렬 클럭(SCL)이라는 두 개의 양 방향 오픈 컬렉터 라인을 사용하여, Master인 PC의 명령을 RS-232 단자를 통해 전달받는다. 이 때, PC에서 Register Control 블록에 설정된 Register Set의 데이터를 변경함으로써, SRAM의 주소 생성범위 제어, 탐색 영역 조절, 탐색 영역내에서 현재 매크로 블록의 이동 방향 조절, 움직임 벡터의 LUT 생성 등의 제어가 외부 사용자에 의해 가능하도록 하였다. 이를 통해, 위에서 언급된 신호를 변경하고자 할 경우에 하드웨어를 구성하는 로직의 변경없이 결과를 바로 확인할 수 있도록 하였다.

Ⅳ. 실험 및 고찰

본문에서 제안된 움직임 추정기 구조는 Verilog HDL (Hardware Description Language)을 이용하여 설계하였고, Modelsim 6.0d에 의해 동작 시뮬레이션(Functional Simulation)을 수행하였다. 그 다음, Synplify 합성툴을 이용하여 게이트 레벨 구조 분석 및 Netlist 추출하여 타이밍 시뮬레이션을 통해 움직임 추정기 구조에 대한 검증을 완료하였다. 시뮬레이션 결과는 그림 11과 같으며, 입력 Sequence로 초당 30 프레임의 순차적인 CIF급 foreman 영상 데이터를 이용하여 움직임 추정을 위한 현재 프레임과 참조 프레임의 데이터로 사용하였다. 현재 프레임을 부호화할 때 사용되는 참조 프레임은 연속적으로 배열된 영상데이터에서 현재 프레임 바로 이전 프레임이 필요로 한다. 따라서 특정한 영상 프레임에 대한 움직임 추정이 끝날 때 마다 순차적인 2 프레임 영상 데이터를 텍스트 파일로 변환하여 FPGA에 내장된 SRAM의 입력데이터로 사용하였고, 탐색 영역은 (-16, -16)~(15, 15)로 설정하였다. 최초 16클럭 동안 MED 블록에 배열되어 있는 16개의 MED_X에 16×16 현재 블록과 이전 블록에 대한 데이터가 입력된다. 즉, 16개의 SRAM에서는 시스템 클럭에 맞춰 동시에 8비트 버스 크기로 16×1 픽셀의 데이터가 출력된다. 따라서 처음으로 블록 정합을 시작하기 위해서는 16클럭 동안은 SRAM으로부터 입력만 받게 된다. 그 다음 클럭 부터는 스캔 방향에 따라 SRAM으로부터 탐색 영역에 대한 16×1 또는 1×16 데이터만을 입력받아 실시간으로 움직임 추정을 수행하게 된다. 움직임 추정기 내부에서 데이터 흐름을 살펴보면, 16클록 동안 데이터를 입력받은 MED블록은 참조 블록에 대한 16개의 4×4 SAD값을 계산하여 출력하고, 이 값을 이용하여 그 다음 17번째 클럭에서 7 가지 블록 크기에 해당하는 41개의 값을 Adder Tree를 통해 만들어 내게 된다. 최소값 계산 블록과 움직임 벡터 생성부에서는 탐색 횟수만큼 입력되는 41개의 값을 이전에 저장된 최소값과 비교하여 최소값을 갱신하게 된다. 따라서 움직임 추정기에서 최초로 출력하는데 걸리는 클럭수는 17이 되고,

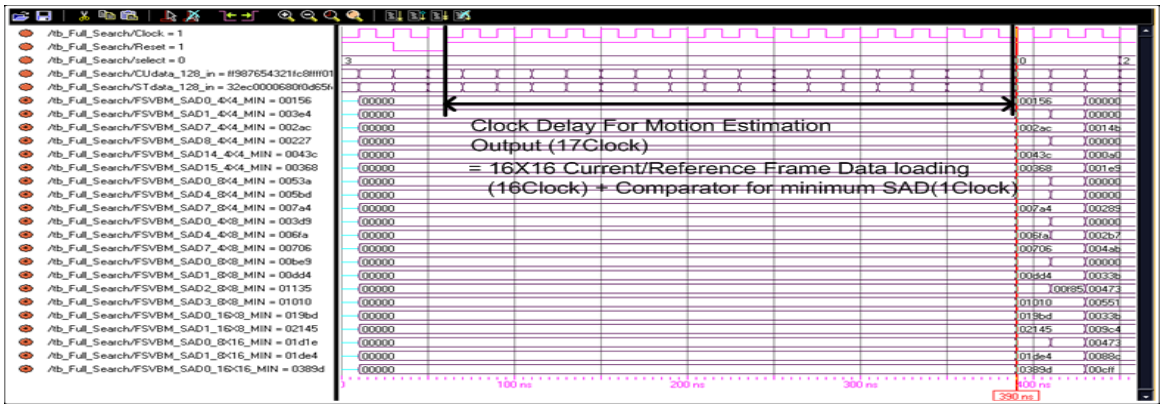

그림 11. 움직임 추정기의 타이밍 시뮬레이션 결과
Fig. 11. The timing simulation result of motion estimator.

전체 탐색 범위가 (-16, -16)~(15, 15)이므로, 현재 블록과 Best Matching되는 참조 블록을 찾는데 필요한 총 클럭 수는 17+(16×16)=273이 된다.

설계된 구조에 대한 모듈별 Delay와 게이트 카운트 수를 비교하기 위하여 FPGA 구현 및 검증을 진행하였다. 사용된 FPGA는 Virtex 4 계열의 XC4VLX200이고, 패키지는 FF1513, Speed 등급은 -11을 적용하였다.

일반적인 구조와 제안된 구조의 Delay를 정확하게 비교하기 위해서 합성 및 P&R(Place & Route) 정보는 동일하게 유지하였다. 제안된 움직임 추정기 구조에는 선계산 기법, Carry Skip Adder 구조, I2C 통신을 위한 인터페이스 블록 등이 적용되었으며, 일반적인 구조는 참고 문헌의 논문[7~8, 11]에서 단위 블록의 병렬 배열을 통한 고속 처리 구조를 사용하였다. 일반적인 구조는 선 계산 방법을 이용한 입력 데이터 예측 방법 대신 전체 입력 비트에 대한 연산을 수행한 후 절대값 및 최소값을 구하는 구조와 Ripple Carry Adder가 적용되었다. 또한 탐색 영역이 고정되어 사용됨으로 I2C 인터페이스 블록이 포함되지 않았다. 표 3과 같이 Delay는 로직에서 차지하는 Gate Delay와 로직의 입·출력을 연결해주는 Net Delay로 구성된다. MED블록에서는 선계산과 CSA 구조를 적용하였으며, 일반적인 구조에 비해 19.89% 정도의 Delay 감소 효과를 얻을 수 있었다. 여기서, 선계산 방법에 RCA 구조를 적용하였을 경우에는 Total Delay가 13.203ns로 제안된 구조의 Total Delay 보다 대략 0.704ns만큼 증가하게 된다. 따라서 CSA구조에 선계산 방법을 결합하여 사용하였을 경우에 Delay 감소 효과를 최대화시킬 수 있다. 41개의 SAD값 계산 블록에서는 CSA 구조를 적용하여 21.97%의 Delay 감소와 최소 SAD값 계산 및 움직임 벡터 생성블록에서는 선계산 기법을 적용하여 3.72%의 Delay 감소를 얻을 수 있었다. 하드웨어의 동작 주파수에 영향을 끼치는 임계 경로는 Delay가 가장 많은 MED 블록에서 나타나며, 일반적인 구조가 64MHz (1/15.602)의 동작 주파수를 갖는 반면에 제안된 구조는 80MHz(1/12.499)의 동작 주파수를 얻을 수 있었다. 이를 통해 초당 30프레임의 CIF 영상을 기존 구조에 비해 고속으로 처리할 수 있게 되었다. 여기서 게이트 수를 분석하면, 일반적인 구조는 게이트 카운트 수가 130.3K이고, 제안된 구조는 142.6K로 9.43% 증가하였다. 게이트 수 증가 요인은 I2C 통신을 위한 인터페이스 회로 추가와 고속의 가산기 사용으로 인해 발생하였다.

표 3. 움직임 추정기의 블록별 Delay 비교.
Table 3. Delay comparison for blocks of motion estimator.

MED	Gate Delay	Net Delay	Total
Conventional	7.124	8.478	15.602
Proposed	5.429	7.070	12.499
% reduction	23.79	16.61	19.89

41 SADs Calculation	Gate Delay	Net Delay	Total
Conventional	8.635	4.575	13.210
Proposed	6.149	4.159	10.308
% reduction	28.79	9.09	21.97

Minimum SAD Calculation & Motion Vector Generation	Gate Delay	Net Delay	Total
Conventional	3.908	0.326	4.234
Proposed	3.774	0.303	4.077
% reduction	3.43	7.06	3.72

V. 결 론

움직임 추정은 영상 부호화 시스템에서 큰 비중을 차지하는 부분으로, 실시간으로 동작하기 위해서는 효율적인 구조를 필요로 한다. 본 논문에서는 많은 연산량을 효율적으로 줄일 수 있도록 병렬 처리를 바탕으로 움직임 추정 감지 블록, 41개의 SAD(Sum of Absolute Difference)값 계산 블록, 최소의 SAD값 계산과 움직임 벡터 생성 블록을 제안하였다. 움직임 추정 감지 블록과 최소의 SAD값 계산기에서는 선계산 방법을 적용하여 입력 Switching Activity를 줄여 고속 구현이 가능하도록 하였으며. 움직임 추정 감지 블록과 41개의 SAD값 계산 블록에서 가장 많은 부분을 차지하는 가산기 구조를 일반적으로 사용되는 Ripple Carry Adder 대신에 Carry Skip Adder를 적용함으로써, Adder Tree 구조를 고속으로 처리할 수 있도록 하였다. 또한 외부에서 탐색 영역 제어와 같은 주요 변수를 쉽게 제어할 수 있도록 하여, 하드웨어 구조의 효율성을 높였다. 시뮬레이션 및 FPGA 검증 결과, 움직임 추정기의 임계 경로를 발생시키는 MED블록에서 일반적인 구조를 적용했을 때보다 19.89%의 Delay 감소 효과를 얻을 수 있었다. 따라서 제안된 움직임 추정기는 고속 처리를 중요한 인자로 사용하는 시스템의 경우에 널리 사용될 수 있을 것이다.

참 고 문 헌

[1] Suk-Ju Kang, Dong-Gon Yoo, Sung-Kyu Lee, and Young Hwan Kim, "Hardware Implementation of Motion Estimation Using a Sub-sampled Block for Frame Rate Up-Conversion," International SoC Design Conference(ISOCC) 2008, pp. 101-104, Nov. 2008.

[2] Chun-Ho Cheung and Lai-Man Po, "A Novel Cross-Diamond Search Algorithm for Fast Block Motion Estimation," IEEE Trans. Circuit and Systems for video technology, Vol. 12, no. 12, December 2002.

[3] B. M. Wang, J. C. Yen and S. Chang, "Zero waiting cycle hierarchical block matching algorithm and its array architectures," IEEE Trans. Circuit and Systems for video technology, Video Technology, Vol. 4, pp. 18-28, Feb. 1994.

[4] R. Srinivasan and K. R. Rao, "Predictive coding based on efficient motion estimation," IEEE Trans. Commun., Vol. COM-33, pp. 888-896, Aug. 1985.

[5] Xiangwen Wang, Jun Sun, Rong Xie, Songyu Yu, and Wenjun Zhang, "An improved block size selection method based on macroblock movement characteristic," multimedia tools and applications, Vol. 43, no. 2, pp. 131-143, May 2009.

[6] Zhou Z, Sun MT and Hsu YF, "Fast variable block-size motion estimation algorithms based on merge and split procedures for H.264/MPEG-4 AVC," ISCAS2004, Vol. 3, pp. 725-728, May 2004.

[7] Siou-Shen Lin, Po-Chih Tseng and Liang-Gee Chen, "Low-power Parallel Tree Architecture For Full Search Block-Matching Motion Estimation," ISCAS2004, Vol. 2, pp. 313-316, May 2004.

[8] Subarna Chatterjee and Amlan Chakrabarti, "Parallel Hardware Design for Motion Estimation," ACEEE 2009 Academy Publisher, International Journal of Recent Trends in Engineering, Vol. 1, no. 1, pp. 653-657, May 2009.

[9] 윤미선, 장승호, 문동선, 신현철, "H.264 동영상 압축에서의 가변 블록과 다중 프레임을 지원하는 효율적인 움직임 추정 방법," 전자공학회 논문지, 제 44권 SD편, 제5호, 58-64쪽, 2007년 5월

[10] Swee Yeow Yap and John V. McCanny, "A VLSI Architecture for Advanced Video Coding Motion Estimation," ASAP'03, Proceedings. IEEE International Conference on, pp. 293-301, June 2003.

[11] Ching-Yeh Chen et al., "Analysis and architecture design of variable block size motion estimation for H.264/AVC," IEEE Trans. Circuit and Systems for video technology, Reg. Paper, Vol. 53, no. 2, pp. 578-593, Feb. 2006.

[12] Jen-Chieh Tuan, Tian-Sheuan Chang, and Chein-Wei Jen, "On the Data Reuse and Memory Bandwidth Analysis for Full-Search Block-Matching VLSI Architecture," *IEEE Trans. Circuit and Systems for video technology*, Vol. 12, no. 1, pp. 61-71, January 2002.

[13] R. Zimmermann and H. Kaeslin, "Cell-Based multilevel Carry-Increment Adders with Minimal AT- and PT-Products," unpublished manuscript, http://www.iis.ee.ethz.ch/~zimmi/

[14] A. Amin, "High-Speed Self-Timed Carry-Skip Adder," *Institution Engineering Technology-IET, IEE Proceedings - Circuit Devices and Systems*, pp. 574-582, Vol. 153, no. 6, December 2006.

[15] 장영범, 오세만, 김비철, 유현중, "H.264 움직임 추정을 위한 효율적인 SAD 프로세서," 전자공학회 논문지, 제 44권 SP편, 제2호, 74-81쪽, 2007년 3월

[16] http://www.semiconductor.philips.com/buses/i2c

저 자 소 개

황 종 희(정회원)

2001년 인하대학교 반도체공학과 학사 졸업.
2005년 인하대학교 정보통신 공학과 석사 졸업.
2009년～현재 연세대학교 전기전자공학과 박사과정
2001년 LG전자 평택 DAV 사업부 입사
2005년～현재 LG Display 선행개발 회로설계 선임 연구원
<주관심분야 : 영상 신호처리, 반도체 SOC>

최 윤 식(정회원)

1979년 연세대학교 전기공학과 학사 졸업
1984년 Case Western Reserve Univ. 시스템공학과 졸업.
1987년 Pennsylvania State Univ. 전기공학과 석사 졸업
1990년 Purdue Univ. 전기공학부 박사 졸업
1990년～1993년 (주)현대전자 산업전자 연구소 책임 연구원
1993년～현재 연세대학교 전기전자공학부 정교수
<주관심분야 : 비디오, 영상 신호처리, HDTV>