論文 98-23-1-4

저 전송률 비디오 코덱을 위한 움직임 예측기의 VLSI 구조

正會員 김 기 현*, 천 대 녕*, 민 병 기*

VLSI Architecture of Motion Estimator for Low bit rate video Codec

K. H. Kim*, D. N. Chun*, B. K. Min* *Regular Members*

요 약

본 논문에서는 저 전송률 비디오 코덱에 적용할 수 있는 최적의 움직임 예측기를 제안한다. 제안하는 움직임 예측기는 선형 세미-시스토릭 어레이 구조에 기초한 완전탐색 블록정합 알고리듬을 사용한다. 메모리 구조는 메모리의 크기를 최소로 하고, 연산기의 효율을 최대로 유지시켜 VLSI로 구현하기에 적합하다. 내부 메모리는 데이터의 중복 사용을 고려하여 내부 메모리에 최소한의 데이터로 초기화한 다음, 연산 수행과 병행하여 연산에 장애를 주지 않으면서 필요한 탐색영역 데이터의 다운로드를 수행 한다. 따라서, 연산기의 효율 증대와 메모리 크기의 최소화, 데이터 입력의 분산은 움직임 예측기의 성능을 향상시켰다. 또한, H. 263 표준의 선택적 추가 기능인 Unrestricted Motion vector mode8 지원한다. 본 논문의 움직임 예측기는 VHDL 설계와 시뮬레이션이 완료되어, 저 전송률 비디오 코덱의 다른 모듈(DCT, Quantizer, RLC, VLC 등)과 통합되었다.

ABSTRACT

In this paper, we proposed a very suitable motion estimator for low bit rate video codec. Proposed motion estimator is based on linear semisystolic arrays architecture using full-search block matching algorithm. Memory architecture for proposed motion estimator is suitable implemented VLSI because of minimized its physical size and maximized the efficiency of PE(processing element). The memory is initiated minimum data used overlap search area data, and than download of data for compute is done concurrently operation compute. As a result, performance of motion estimator is enhanced to maximized the efficiency of PE and minimized of memory size, distributed of data input. In addition, proposed motion estimator support Unrestricted Motion vector mode which is optional in H.263 standard. We finished the design of this ME in VHDL and simulation, now integrated other modules of low bit rate video codec.

*한국전자통신연구원
論文番號:97383-1024
接受日字:1997年 10月 24日

Ⅰ. 서 론

저 전송률 비디오 압축 방식인 H.263표준은 표준화 작업 초기에는 CIF(352×288 화소) 화면의 1/4인 QCIF 화면 이하의 저 화질에 대한 고 압축 개념으로 64Kbps 이내의 대역폭을 갖는 응용으로 제한하였다. 그러나 최근 H. 263의 표준 안에는 상기 다양한 대역폭을 제공하는 통신망에 따른 멀티미디어 통신 단말 표준들을 포괄적으로 지원하기 위하여 Sub-QCIF, QCIF, CIF, 4CIF, 16CIF 크기의 화면을 지원하도록 하였다[1].

이와 같은 동영상 데이터의 압축은 동영상의 전송을 위해서는 필수적인 요소이며, 또한 전체 압축율의 많은 부분을 차지하는 움직임 예측/보상(ME:Motion Estimation/MC:Motion Compensation)에 의한 압축 방법은 매우 중요하다[1]. 움직임 예측 방법에는 전체 화면을 몇 개의 블록으로 나누어 블록 단위로 이동 벡터(MV:Motion Vector)를 찾아내는 블록정합 알고리듬(BMA:Block Matching Algorithm)이 가장 널리 알려져 있다. 특히, 탐색영역의 모든 후보 블록을 대상으로 블록정합을 수행하는 완전탐색 블록정합 알고리듬(FBMA:Full-search Block Matching Algorithm)은 정확도에서 가장 우수하며 데이터의 공급이 용이하여 널리 사용되고 있다[3~9].

완전탐색 블록정합 알고리듬은 많은 계산이 필요로 하여 실시간 처리를 위한 여러 가지 특별한 하드웨어 구조가 제안되었다. 이 제안된 구조들은 실시간 처리를 위해 요구되어지는 계산량과 입출력 핀의 갯수를 줄이면서 메모리의 대역폭을 최소화하는 문제, VLSI로 구현할 경우 비용의 최소화 등을 고려하여 설계, 제안되었다. 그러나 이들 문제의 동시 해결은 많은 어려움이 있어 연산기(PE:Processor Element)의 갯수는 탐색범위의 크기나 데이터 전송률에 좌우된다. 또한, 비록 데이터 입력 대역폭을 줄이지만, 기준 블록과 탐색영역의 데이터 초기화에 많은 시간을 소요하거나, 움직임 예측기 성능을 저하시키는 경우가 발생한다. 예를 들면 16×16 기준블록과 [-16~15]의 탐색영역 데이터를 초기화 시키는데 소요되는 시간이 전체 연산시간의 반 이상을 차지하는 경우가 발생한다[6][7]. 그리고 가장자리의 기준블록에 대한 탐색을 할 경우 어떤 연산기들은 동작하지 않아 전체 움직임 예측기의 성능을 저하시키는 경우도 있다[5].

이미 제안된 많은 움직임 예측기는 작게는 8개에서 많게는 1024개 이상의 연산기를 가지고 있다. 물론 각각의 응용 환경과 동작 속도에 따라 그 갯수는 정해진다. 이미 제안된 다른 움직임 예측기들의 구조를 크게 한 개의 연산기에서 한 후보 블록에 대한 연산이 모두 이루어지도록 하는 것과 몇 개의 연산기가 한 그룹이 되어 한 후보 블록에 연산을 수행하는 것으로 나눌 수 있다[2].

첫 번째 경우의 대표적인 구조로 Yang의 시스템이 있다[3]. Yang의 구조로 구성할 경우 연산기는 기본적으로 32개가 필요하며, 기준 블록의 크기 변화에 따라 시스템 구성이 용이하다. 그러나 탐색 범위의 변화에 따라 연산기의 효율을 최대로 유지할 수 없는 단점을 가지고 있다. 변형된 방법으로 한 연산기에서 두 개의 후보 블록에 대한 연산을 하도록 연산기를 설계한 시스템도 있다[9]. 그러나 이러한 구조는 탐색영역 데이터가 순차적으로 연산기 어레이로 입력이 이루어지지 못하고, 중복 사용되는 데이터를 이용하기 위하여 지연 버퍼(Delay Buffer)를 이용함으로써 하드웨어의 추가비용이 발생한다. 또한 이 구조 역시 탐색 범위의 변화에 대해 융통성을 가지고 있지 않다. 탐색 영역 -8~7인 경우 각 연산기에 있는 계산되고 있는 SAD 값을 저장하는 두 개의 레지스터중 한 개는 사용하지 않게 되어 전체적인 움직임 예측기의 효율이 떨어진다.

본 논문에서는 두 번째 방법을 이용하여 움직임 예측기를 설계한다. 데이터의 중복 이용도가 높고, 연산기의 효율이 탐색영역의 위치나 탐색범위에 따라 변하지않는 시스템을 제안한다.

이러한 관점에서 본 논문에서는 저 전송률 비디오 코덱에 가장 적합한 움직임 예측기의 설계를 위해서 완전탐색 블록정합 알고리듬을 사용하고, 아래의 사항에 대해 분석하여 최적의 움직임 예측기의 VLSI 구조를 제안하고자 한다.

1) 연산기의 갯수 및 연산 동안 연산기의 효율.
2) 칩의 크기와 데이터 입력 대역폭을 결정하는 내부 메모리의 크기.
3) 데이터 입력 대역폭과 입출력 핀의 갯수

본 논문은 2장에서 완전탐색 블록정합 알고리듬에 대해 간략하게 설명하고, 움직임 예측기의 처리 계산량에 대해 기술한다. 3장에서는 제안하는 움직임 예측기의 연산기 구조, 선형 세미-시스토릭 어레이 구조 및 동작 상태를 데이터 흐름도를 통하여 설명한다. 4장에서는 제안한 움직임 예측기의 탐색 범위의 변화에 따른 융통성과 추가된 다른 기능들에 대해서 설명한다. 5장에서는 데이터 입력을 위한 최적의 내부 메모리의 크기를 결정하고, 구조 및 입력 동작에 대해서 설명한다. 6장에서는 성능 평가 및 시뮬레이션 결과와 논리 합성(Logic Synthesis)된 회로를 설명한다.

Ⅱ. 완전탐색 블록정합 알고리듬

그림 1은 완전탐색 블록정합 알고리듬을 나타내고 있다. 그림 1에서 현재 화면은 크기가 N×N인 기준 블록 단위로 나누어진다. 나누어진 기준블록 행렬에서 한 기준 블록 행(row)을 슬라이스(Slice)라 한다. 이전 화면에서 기준블록과 같은 위치에서 수평, 수직 범위 −p~p−1인 탐색 범위내의 영역을 탐색영역이라 한다.

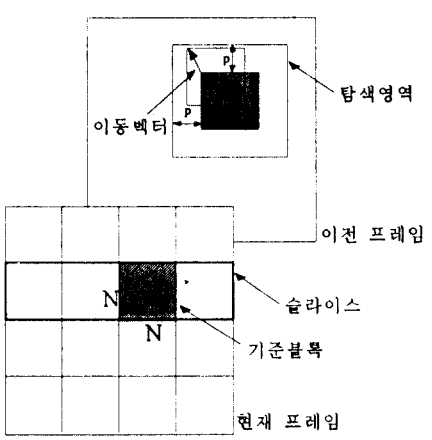


그림 1. 완전탐색 블록정합 알고리듬
Fig. 1 Full-search Block Matching Algorithm

탐색영역내의 후보 블록들 중 가장 작은 SAD(Sum of Absolute Difference)값을 갖는 블록의 이동벡터가 기준블록의 이동벡터가 된다.

그림 1의 완전탐색 블록정합 알고리듬을 수식으로 표현하면 식 (1)과 같다.

$$SAD(u, v) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |r(i, j) - s(u+i, v+j)|$$

$$MV = (u, v)_{\min(SAD(u, v))} \qquad \text{식 (1)}$$

$$-p \le u < p$$

$$-p \le v < p$$

식 (1)에서 $r(i, j)$는 기준블록 데이터이며, $s(u+i, v+j)$는 후보블록의 데이터이다. 한 이동벡터 MV는 탐색영역내의 가능한 모든 후보블록의 이동벡터 (u, v)중에서 가장 작은 SAD값을 갖는 벡터가 된다.

식 (1)을 통하여 완전탐색 블록정합 알고리듬의 계산량을 예측할 수 있다. 예로써 16×16의 기준블록과 [−16~15]의 탐색 범위를 가질 경우 한 기준 블록의 이동 벡터를 구하기 위해서 2^{18} AD(Absolute Difference)연산이 필요하며, QCIF 30 프레임을 실시간 처리하기위해서는 1초당 약 6×10^8의 AD 연산이 이루어져야 한다. 완전탐색 블록정합 알고리듬은 정확도가 높은 반면 계산량이 많기 때문에 하나의 연산기로서는 실시간 처리가 불가능하며 여러 개의 연산기를 사용하는 병렬처리가 필요하다. [2] 또한 한 개의 이동 벡터를 구하기 위해서 메모리의 대역폭과 전체 계산 시간을 줄이는 것은 움직임 예측기의 설계에 있어 매우 중요한 요소가 된다.

Ⅲ. 제안된 움직임 예측기 구조

본 장에서는 연산기 구조, 선형 세미-시스토릭 어레이 구조를 설명하고, 연산기의 동작 상태는 데이터 흐름도를 통하여 설명한다.

그림 2 (a)에는 기준블록이 나타나 있으며, (b)에 탐색영역 데이터가 나타나 있다. 그림 2 (a)에서 R0는 기준 블록의 첫번째 행이고, S0, S16은 각각 탐색영역의 첫번째와 열 일곱번째 행이다. 이동 벡터를 (u, v)로 표현할 경우 'u' 값을 같이 갖는 후보 블록의 영역을 "탐색영역 슬라이스"라 칭하고, 그림 2 (b)에 잘 나타나 있다. 그림 2 (b)의 예에서는 'u'의 값이 −16인 탐색영역 슬라이스가 첫번째 탐색영역 슬라이스

가 되며, -15에 대한 탐색영역 슬라이스는 두번째 탐색영역 슬라이스가 된다. 두개의 이웃한 탐색영역 슬라이스는 두개의 탐색영역 행, S0와 S16을 제외하고는 중복된 값을 가지며, 그림 2 (b)에 빗금 친 부분으로 나타나 있다.

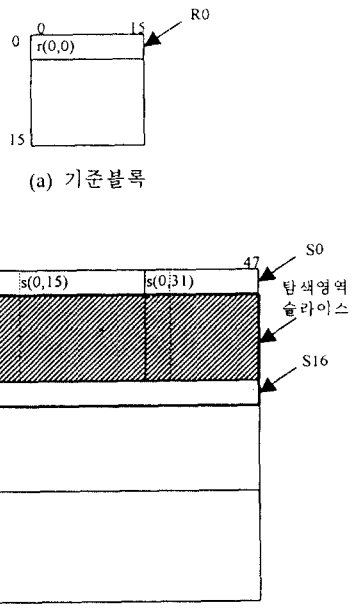


그림 2. 기준블록과 탐색영역
Fig. 2 Reference block and Search area

그림 2 (b)에서 s(0, 0)는 이전화면에서 기준블록 (a)에 대해 (-16, -16)의 이동벡터를 가지는 후보블록의 좌측 상단에 있는 화소(pixel)이다. 그림 2 (b)에서 화소 s(0, 0)는 SAD(-16, -16)의 AD 연산 |r(0, 0)-s(0, 0)|에 한번 사용되고, s(0, 1)은 SAD(-16, -16)의 AD 연산 |r(0, 0)-s(0, 1)|와 SAD(-16, -15)의 AD 연산 |r(0, 1)-s(0, 1)|에 두번 사용되며, s(0, 15)~s(0, 31) 화소는 16번 사용된다. 결국 s(0, 15) 값이 내부 연산기에 입력되었을 때 s(0, 15)와 연산 가능한 SAD(-16, -16), SAD(-16, -15), ..., SAD(-16, -1)의 AD 연산이 각각의 연산기에서 수행되며, s(0, 31)이 입력되었을 때는 SAD(-16, 0), SAD(-16, 1), ..., SAD(-16, 15)의 AD 연산이 수행될 수 있다. 이렇게 할 경우 한 기준블록에 대한 탐색영역 데이터의 중복사용을 최대로 할 수 있다. 이렇게 연산이 이루어지게 하기위해서는 각 연산기에는 하나의 기준블록 데이터가 일정한 시간동안 기억되어 있어야 한다. 즉 r(0, 0)는 s(0, 31)과의 AD 연산이 이루어질 때까지 한 개의 연산기에 기억되어 있어야 하며, r(0, 15)는 s(0, 47)과의 AD 연산이 이루어질 때까지 또 다른 연산기에 기억되어 있어야 한다.

그림 3(a)는 본 논문에서 제안하는 연산기가 나타나 있고, 그림 3 (b)는 본 논문에서 제안하는 선형 세미-시스토릭 어레이 구조이다. 그림 (a)의 R_path는 기준블록 데이터를 공급하는 경로이고, S0_path와 S1_path는 탐색영역 데이터 공급을 위한 경로이다. 그림 3 (a)에서 Reg은 레지스터이고, Mux은 멀티플렉서(multiplexor)이며, AD(Absolute Difference)는 입력된 두값 차이의 절대값을 계산하는 부분이다. 각 연산기에는 기준 블록의 데이터를 일정기간 저장하는 레지스터와 제어 신호에 의해 선별된 탐색영역 데이터가 저장되는 레지스터가 있다. 또한 AD 연산 결과를 저장하는 레지스터와 이웃한 왼쪽 연산기(PE_{i-1})에서 전달되는 부분 합(Psum:Partial Sum)의 결과에 자신의 연산 결과를 다시 더한 결과를 저장하는 레지스터가 있다. 각 연산기에는 제어신호를 저장, 다음 연산기로 전달하는 레지스터가 있으며, 제어신호는 그림 3(a)에 점선으로 나타나 있다. 모든 제어 신호는 왼쪽 연산기(PE_{i-1})로부터 입력되어 오른쪽 연산기(PE_{i+1})로 전달되는 파이프라인으로 되어 있다. 본 논문에서 제안하는 연산기에는 기준 블록의 값을 저장하는 시점을 알리는 제어 신호와 탐색영역 데이터의 경로를 결정하는 제어 신호, 두개의 제어 신호 만으로 동작함으로써 제어기의 구조가 간단하여 VLSI로 구현하기가 쉽다.

그림 3 (b)의 연산기들은 3개의 경로를 통하여 기준블록 데이터와 탐색영역 데이터를 공급 받고, 각각의 연산기는 부분 합을 전달하는 경로로 연결되어 있다. 마지막 연산기(PE15)의 오른쪽에는 PE15에서 출력되는 부분 합을 누적하는 "부분 합 덧셈기(Partial Sum Adder)"와 SAD 연산이 완료되었을 경우 서로 비교하여 가장 작은 SAD 값을 찾는 "비교기(Comparator)"가 있다.

본 장의 나머지 부분에서는 그림 4을 이용하여 제

안된 선형 세미-시스토릭 어레이 구조의 동작을 설명한다. 그림 4의 각 연산기의 열에는 매 클럭마다 연산기에서 이루어지는 AD연산이 나타나 있다. R열에는 각 연산기에 공급되는 기준블록 데이터가 나타나 있고, S0와 S1에는 탐색영역 데이터가 나타나 있다.

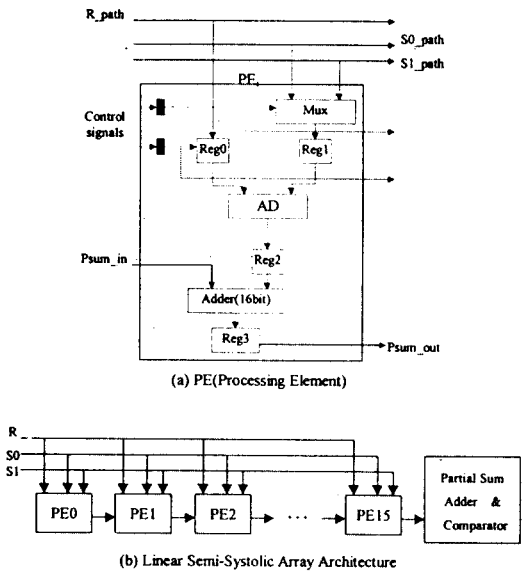


그림 3. 연산기와 선형 세미-시스토릭 어레이 구조의 블록도
Fig. 3 Block Diagram of Processing element and Linear Semi-systolic Array System

T=0에서 기준블록의 첫 번째 행, R0의 첫 번째 데이터 r(0, 0)는 PE0의 Reg0에 저장되고, 탐색영역의 첫번째 행, S0의 첫 데이터 s(0, 0)가 입력되어 PE0의 Reg1에 저장되어 AD연산이 이루어진다. T=1일때 PE1에는 r(0, 1)이 저장되며, s(0, 1)은 PE0, PE1에 같이 저장되어 AD연산이 이루어지며, T=0일때 PE0에서 연산된 결과는 T=1에서 PE1으로 전달되어 PE1의 연산 결과와 더해져 T=2 일때 PE2로 전달된다. 결과적으로 PE0는 한 탐색영역 슬라이스 내에 있는 모든 후보 블록의 한 행에 대한 부분 합에 있어 첫번째 화소의 AD 연산만을 하고, PE1은 두 번째 화소의 AD 연산을 하고 PE0의 결과와 더해서 두번째 화소까지의 부분 합을 계산한다. 같은 방법으로 PE15는 어떤 탐색영역 슬라이스 내에 있는 모든 후보 블록의 어떤 한 행에 대한 16번째 화소까지의 부분 합을 계산하게 된다. 그림 4의 T=15일때 PE15에서는 첫번째 탐색영역 슬라이스의 첫번째 후보 블록의 첫번째 행의 마지막 AD연산이 이루어지며, T=16에 첫번째 탐색영역 슬라이스의 첫번째 후보 블록의 첫번째 행의 부분 합이 PE15에서 출력된다. T=16부터 T=2×16+15까지 첫번째 탐색영역 슬라이스 32개의 후보 블록의 첫번째 행의 부분 합이 출력된다. 출력된 부분 합은 그림 3 (b) 부분 합 덧셈기(Partial Sum Adder) 모듈의 저장 레지스터에 각각 저장된다. 식 (2)는 식 (1)의 SAD 연산에서 한 행에 대한 부분 합을 SAD'으로 표현한 것이다.

$$SAD(u, v) = \sum_{i=0}^{15} SAD'(u, v, i)$$

$$SAD'(u, v, i) = \sum_{j=0}^{15} |r(i, j) - s(u+i+16, v+j+16)| \quad \text{식 (2)}$$

$$-16 \le u < 16$$

$$-16 \le v < 16$$

식 (2)에서 u, v는 탐색 범위 −16~15 사이의 정수값이 되고, 탐색영역 데이터 $s(u+I+16, v+j+16)$에서 식 (1)과 비교하여 좌표에 16을 더한 것은 탐색영역의 가장 좌측 상단의 화소의 절대 위치 값을 s(0, 0)로 한 이유에서다. 앞 절에서 설명한 연산과정을 식 (2)를 적용해서 표현하면 T=16부터 T=2×16+15까지 SAD'(−16, −16, 0), SAD'(−16, −15, 0), ..., SAD'(−16, 15, 0)가 출력되는 것이다. 연산기의 효율을 100%로 하기위해서는 T=2×16+0에서 두 번째 행에 대한 부분 합 연산이 PE0에서 시작되어야 한다. 기준 블록 데이터는 R_path를 통하여 공급될 수 있지만, 탐색영역 데이터는 S1_path가 없을 경우 사용되고 있는 S0_path를 통하여 데이터를 공급할 수 없게되어 연산기의 효율을 100%가 될 수 없다. 연산기의 효율을 100%로 하기위해 또 다른 탐색영역 데이터 공급 경로 S1_path가 필요로 하게 된다. T=3×16+0에서 두 번째 행에 대한 부분 합 결과가 PE15을 통하여 출력되면, 부분 합 덧셈기 모듈에 저장되어 있는 첫 행에 대한 부분 합과 더해져 다시 저장된다. 즉 부분 합 덧셈기에 저장되어 있는 SAD'(−16, −16, 0) 값은 SAD'(−16, −16, 1)과 더해져 저장되

Time	R	S0	S1	PE0	PE1	PE2	. . .	PE15
0	r(0,0)	s(0,0)		|r(0,0)-s(0,0)|				
1	r(0,1)	s(0,1)		|r(0,0)-s(0,1)|	|r(0,1)-s(0,1)|			
2	r(0,2)	s(0,2)		|r(0,0)-s(0,2)|	|r(0,1)-s(0,2)|	|r(0,2)-s(0,2)|		
14	r(0,14)	s(0,14)		|r(0,0)-s(0,14)|				
15	r(0,15)	s(0,15)		|r(0,0)-s(0,15)|	|r(0,1)-s(0,15)|			|r(0,15)-s(0,15)|
16+0		s(0,16)		|r(0,0)-s(0,16)|	|r(0,1)-s(0,16)|	|r(0,2)-s(0,16)|		|r(0,15)-s(0,16)|
16+1		s(0,17)		|r(0,0)-s(0,17)|	|r(0,1)-s(0,17)|	|r(0,2)-s(0,17)|		
						|r(0,2)-s(0,18)|		
								|r(0,15)-s(0,30)|
16+15		s(0,31)		|r(0,0)-s(0,31)|				|r(0,15)-s(0,31)|
2x16+0	r(1,0)	s(0,32)	s(1,0)	|r(1,0)-s(1,0)|	|r(0,1)-s(0,32)|			|r(0,15)-s(0,32)|
2x16+1	r(1,1)	s(0,33)	s(1,1)	|r(1,0)-s(1,1)|	|r(1,1)-s(1,1)|	|r(0,2)-s(0,33)|		
						|r(1,2)-s(1,2)|		
2x16+14		s(0,46)	s(1,14)					|r(0,15)-s(0,46)|
2x16+15	r(1,15)		s(1,15)	|r(1,0)-s(1,15)|	|r(1,1)-s(1,15)|			|r(1,15)-s(1,15)|
					|r(1,1)-s(1,16)|			
						|r(1,2)-s(1,17)|		
				:				
				:		. . . . .		
29x16+15		s(14,31)						
30x16+0	r(15,0)	s(14,32)	s(15,0)	|r(15,0)-s(15,0)|	|r(14,1)-s(14,32)|			
	r(15,1)	s(14,33)	s(15,1)	|r(15,0)-s(15,1)|	|r(15,1)-s(15,1)|	|r(14,2)-s(14,33)|		
						|r(15,2)-s(15,2)|		
								|r(14,15)-s(14,46)|
30x16+15	r(15,15)		s(15,15)	|r(15,0)-s(15,15)|				|r(15,15)-s(15,15)|
31x16+0			s(15,16)	|r(15,0)-s(15,16)|	|r(15,1)-s(15,16)|			|r(15,15)-s(15,16)|
					|r(15,1)-s(15,17)|	|r(15,2)-s(15,17)|		
						|r(15,2)-s(15,18)|		
			s(15,31)	|r(15,0)-s(15,31)|				|r(15,15)-s(15,31)|
32x16+0	r(0,0)	s(1,0)	s(15,32)	|r(0,0)-s(1,0)|	|r(15,1)-s(15,32)|			
32x16+1	r(0,1)	s(1,2)	s(15,33)	|r(0,0)-s(1,1)|	|r(0,1)-s(1,1)|	|r(15,2)-s(15,33)|		
						|r(0,2)-s(1,2)|		
		s(1,14)	s(15,46)	|r(0,0)-s(1,14)|				|r(15,15)-s(15,46)|
32x16+15	r(0,15)	s(1,15)		|r(0,0)-s(1,15)|				|r(0,15)-s(1,15)|

그림 4. 선형 세미-시스토릭 어레이 시스템의 데이터 흐름도.
Fig. 4 Data flow of the Linear Semi-systolic Array System

고, T=4×16+15에서 SAD'(−16, 15, 0)과 SAD'(−16, 15, 1)이 더해져 저장 된다.

이와 같은 연산이 반복 되어 T=30×16+0에서 첫 번째 탐색영역 슬라이스의 마지막 행에 대한 부분 합 연산이 PE0에서 시작되며, T=31×16+0부터 첫 번째 탐색영역 슬라이스 내의 모든 후보 블록에 대한 SAD 연산이 완료된다.

SAD(−16, −16), SAD(−16, −15), ..., SAD(−16, 15) 값들은 순서적으로 계산이 완료되면서 비교되어 SAD(−16, 15) 값이 계산이 완료되어 비교기에 입력되면 첫번째 탐색영역 슬라이스 내에 있는 후보 블록 중에서 가장 작은 SAD 값과 해당 벡터 값만 비교기에 저장된다.

T=32×16+0에서는 S1~S16으로 구성된 두 번째 탐색영역 슬라이스에 대한 첫번째 행의 부분 합 연산을 위해 R0와 S1이 R_path과 S0_path를 통하여 입력되며, 또 다음 탐색영역 슬라이스에 대해도 같은 방법으로 계속해서 연산이 이루어진다.

그림 4를 통하여 첫번째 탐색영역 슬라이스에 대한 *SAD'* 연산이 이루어지는데 소요되는 시간을 수식으로 표현하면 식 (3)과 같다.

$$T_{slice} = T_{init} + T_{SAD'}$$
$$T_{SAD'} = T_{row} \times N \qquad \text{식 (3)}$$

식 (3)에서 T_{init}은 16으로 처음 16개의 기준블록 데

이터가 각각의 연산기에 저장되면서 연산이 이루어지는 초기화에 소요 시간이고, $T_{SAD'}$은 한 개의 탐색영역 슬라이스 연산에서 부분합이 마지막 연산기에서 출력되는 시간을 나타낸다. 식 (3)에서 T_{row}은 32의 값을 가지며, 한 개의 탐색영역 데이터 행이 입력될 경우 동시에 계산이 이루어지는 후보 블록의 갯수가 된다. 그리고, 곱해지는 N은 16으로 기준블록 행의 갯수가 된다. 식 (3)을 이용하여 한 개의 기준블록에 소요되는 총 시간을 수식으로 표현하면 식 (4)와 같다.

$$T_{bock} = T_{init} + T_{SAD'} \times SR \qquad \text{식 (4)}$$

식 (4)에서 SR(Search Range)은 탐색범위의 크기를 나타내고, 같은 의미로 탐색영역 슬라이스의 갯수와 같다. 본 예에서는 32가 된다. 식 (4)를 통하여 16×16의 기준블록과 −16~15의 탐색범위를 갖는 예에서 완전탐색 블록정합이 이루어지는데 소요되는 시간은 16,400 사이클이 된다.

Ⅳ. 탐색 범위의 융통성을 갖는 이동 예측기

본 논문에서 제안한 움직임 예측기는 탐색영역 −8~7를 만족하도록 설계되어 있다. 그 연산의 데이터 흐름을 그림 5에 잘 나타나 있다. S16은 S0가 연산기에 입력된 뒤부터 첫번째 탐색영역 슬라이스의 연산이 끝날 때까지 초기화가 이루어진 다음 두번째 탐색영역 슬라이스에 대한 연산이 이어진다. 그림 5에서와 같이 탐색영역의 변화에도 본 논문에서 제안하는 움직임 예측기의 연산기 효율이 100%가 됨을 알 수 있다. 본 논문의 서론에서 가장 자리에 있는 기준블록에 대한 탐색 범위의 변화에 따른 연산기의 효율 저하 문제를 기술하였다. 예로써 QCIF의 경우 첫번째 기준블록 슬라이스의 첫번째 기준 블록은 벡터 u, v의 범위가 0~15가 되며, 연속되는 9개의 기준블록의 u 벡터 값은 −16~15, v 벡터 값은 0~15가 되며 마지막 기준블록도 변하게 된다. Yang의 알고리듬은 위와 같은 경우 −16~15의 경우를 제외한 나머지 경우에 대해서는 연산기의 효율이 저하됨을 알 수 있다. 그러나 본 논문에서 제안하는 선형 세미-시스토릭 어레이 구조는 2장과 앞 절에서 설명한 것과 같이 탐색범위의 변화에 능동적으로 대처해 기준블록의 크기가 16×16으로 정해진 경우에 있어서는 어떠한 탐색범위에서도 연산기의 효율이 100%로 유지 된다.

그림 5와 같은 경우 한 개의 기준블록 연산에 소요되는 시간을 식 (4)를 이용하여 계산하면 식 (5)와 같다.

$$T_{bock} = 16 + (16 \times 16) \times 16 = 4112 \qquad \text{식 (5)}$$

식 (5)을 통하여 16×16 기준블록과 벡터 'u'와 'v'의 범위가 0~15 인 탐색영역에 대한 완전탐색 블록정

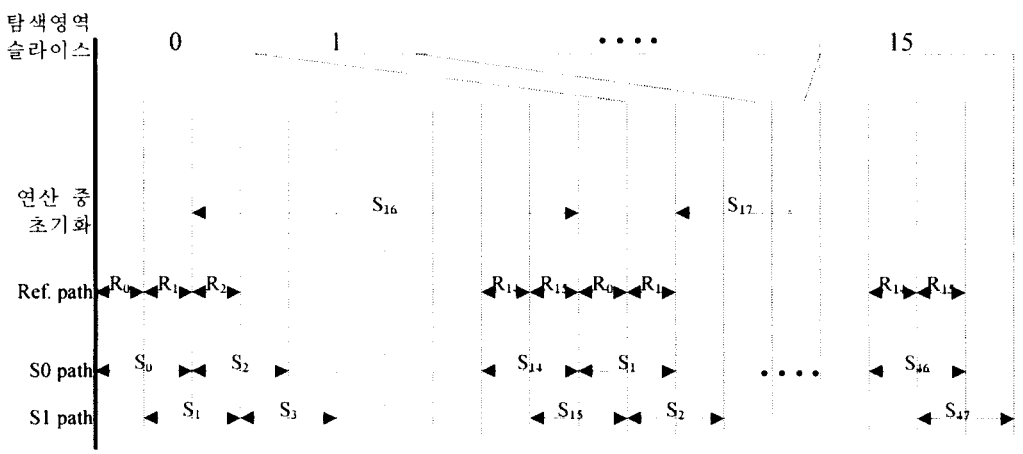


그림 5. 탐색범위가 −8~7 일때 데이터 흐름도.
Fig. 5 Data Flow Diagram for Search range −8~7.

합에 소요되는 시간은 4,112 사이클이 된다. 식 (5)과 같은 연산은 한 프레임 모서리에 위치한 4개의 기준 블록에서 이루어지거나, 탐색범위를 −8~7인 경우 가장자리에 있지 않은 기준블록의 연산에서 나온다. 기준블록 위치 또는 실질적인 탐색범위의 변화에 따라 탐색범위가 변하며, 그 기준블록의 이동 벡터를 구하는데 소요되는 시간은 식 (4)에 잘 나타나 있다.

식 (4)에서 병렬로 연산되는 후보블록의 수는 벡터 (u, v)에서 'u'에 의해 영향을 받고, 탐색영역 슬라이스 갯수는 'v' 값에 영향을 받는다. 또한 식 (4)는 16개의 연산기로 구성된 선형 세미-시스토릭 어레이와 기준블록의 크기가 16×16인 환경에서 유도한 식이다.

V. 최적의 내부 메모리 환경

본 장에서는 VLSI로 제안된 움직임 예측기를 구현할 경우 넓은 면적을 차지하는 내부 메모리의 크기와, 데이터의 초기화 및 필요한 데이터의 외부 메모리로부터 다운로드 과정을 설명한다.

먼저 기준 블록 데이터는 그림 5에서와 같이 매 탐색영역 슬라이스를 연산할 경우 사용되므로 한 데이터는 32번 필요로 한다. 이와 같은 결과에서 기준블록 데이터는 내부 메모리에 모두 저장하는 것이 외부 메모리와의 입력 대역폭을 줄이는데 유리하여, 그 크기는 16×16 byte이다. 탐색영역 데이터를 위한 내부 메모리의 크기를 결정하기 위한 자료는 표 1에 나타나 있다.

표 1. 두개의 메모리 크기에 대한 비교 (단위 : byte)

메모리 크기	초기화 1	초기화 2	중복 영역	연산 중 입력 영역
16×47	16×47	16×47	15×47	32×47
47×47	47×47	16×47	16×47	0

표 1에서 초기화 1은 기준블록 슬라이스에서 처음 기준블록을 위해 필요한 탐색영역 데이터의 크기를 나타내며, 초기화 2는 첫번째 기준블록 이후 연속되는 기준블록 간에 필요한 탐색영역 데이터의 크기를 나타낸 것이다. 중복 영역은 한 기준 블록의 SAD 연산에 있어 한 탐색영역 슬라이스를 마치고 다음 탐색영역 슬라이스에 대해 중복되는 데이터의 크기이며, 이와 같은 경우 한 기준블록의 SAD 연산 중에 발생하는 외부 메모리에서 입력해야 할 데이터의 크기는 표 1의 연산 중 입력 영역에 나타나 있다.

표 1에서 메모리 크기 16×47 bytes의 연산 중 입력 영역의 데이터를 연산기의 효율이 떨어지지 않게 입력 하도록 설계하면, 47×47 byte의 메모리와 같은 성능을 가지며, 메모리의 크기를 2/3 줄일 수 있다. 이러한 문제를 해결하기 위해서 본 논문의 탐색영역 내부 메모리는 Two-Port RAM으로 설계되어 있다. Two-Port RAM은 읽기와 쓰기 부분이 분리되어 있어 어떤 주소의 데이터를 읽는 중 다른 주소에 쓰기를 할 수 있다.

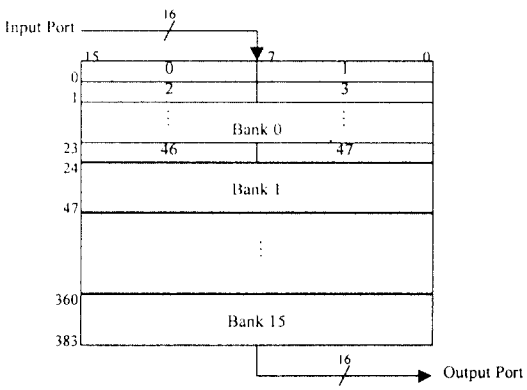


그림 6. 탐색영역 메모리
Fig. 6 Memory for Search Area

그림 6에는 탐색영역 데이터를 저장하는 메모리가 나타나 있다. 입력부와 출력부가 분리되어 있어 같은 시점에 서로 다른 주소에 쓰기와 읽기가 가능하다. 본 메모리는 16개의 뱅크(Bank)로 이루어져 있으며, 데이터 폭은 16비트이고, 각각의 뱅크는 16비트 데이터 24개, 즉 48개의 8비트 데이터를 저장한다. 16비트의 데이터 폭을 가진 메모리는 한번의 메모리 읽기나 쓰기에서 두개의 화소 데이터를 처리할 수 있다. 이와 같은 설계는 두개의 데이터 경로에 매 클럭 순차적인 탐색영역 데이터 공급이 가능하게 한다. 한 개의 뱅크에는 한 행의 탐색영역 데이터가 저장되며, 한 뱅크에 저장되는 탐색영역 데이터의 갯수는 탐색범위의 변화에 따라 변한다.

그림 4에서 SAD 연산을 위해 첫번째 탐색영역 슬라이스 데이터로 초기화된 탐색영역 메모리는 Bank0에는 S0가 Bank1에는 S1이 저장되며, Bank15에는 S15가 저장된다. Bank0의 S0가 연산기에 입력이 완료된 이후, Bank0에 기억되어 있는 S0는 다시 사용되지 않는다. 따라서 Bank0에 두 번째 탐색영역 슬라이스의 연산에 필요한 S16 데이터를 외부 메모리로부터 다운로드 받는다. 첫번째 탐색영역 슬라이스 SAD 연산에 사용되어진 데이터 중 Bank0를 제외한 나머지 데이터는 두번째 탐색영역 슬라이스에 대한 SAD연산에 중복하여 사용됨을 3장에서 보였다. 두번째 탐색영역 슬라이스에 대한 SAD연산은 Bank1에 저장되어 있는 S1을 시작으로 연산기에 입력되며, 마지막으로 Bank0에 저장되어 있는 S16이 입력된다. 또한 두 번째 탐색영역 슬라이스에 대한 연산이 시작되어 Bank1의 S1이 연산기에 입력이 된후 Bank1에는 세 번째 탐색영역 슬라이스의 연산에 필요한 S17을 외부 메모리로부터 다운로드 받는다. 이러한 외부 메모리로부터의 다운로드 과정은 SAD 연산과 병행하여 이루어진다.

이와 같은 방법은 연산 중에 입력해야 할 데이터를 한 개의 행으로 나누어 입력함으로써 외부 메모리에서 한번에 입력 할 경우 발생하는 입력 대역폭에 대한 오버헤드를 줄이고, 하드웨어의 크기를 줄이면서 연산기의 효율은 100%로 유지한다.

첫번째 Bank의 탐색영역 데이터가 연산에 사용된 뒤 나머지 15개 행에 대해 연산이 이루어지는 시간은 다음 탐색영역 슬라이스에 사용되어질 새로운 한 개의 행을 외부 메모리로부터 다운로드 받는데 소요되는 시간으로 충분하다는 것이 시뮬레이션을 통하여 증명되었다.

Ⅵ. 시뮬레이션 결과 및 VLSI 구현

본 논문에서 제안하는 시스템은 VHDL(VHSIC(Very High Scale Integrate Circuit) Hardware Description Language)을 이용하여 설계되고, 시높시스 시뮬레이터(Synopsys Simulator)를 통하여 그 동작이 검증 되었다.

QCIF(176×144 화소) 환경에서 기준블록의 위치에 따른 탐색 범위와 각각의 경우 식 (7)에 의해 계산된 연산에 소요된 클럭 사이클 결과가 표 2에 나타나 있다.

표 2. 탐색범위에 따른 연산 소요 시간

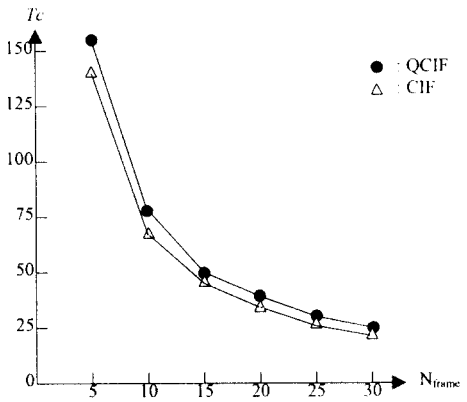
탐색 범위		소요되는 사이클	해당 기준 블록의 갯수	소요 총 사이클
u:0~15	v:0~15	16 +(16×16)×16 = 4,112	4	16,448
u:-16~15	v:0~15	16 +(32×16)×16 = 8,208	18	147,744
u:0~15	v:-16~15	16 +(16×16)×32 = 8,208	14	114,912
u:-16~15	v:-16~15	16 +(32×16)×32 = 16,400	63	1,033,200
전체 계산량				1,312,304

QCIF 한 개의 프레임을 처리하는데 소요되는 사이클 수는 표 2에 의해 1,312,304 cycles이 된다. 결과적으로 QCIF 프레임을 실시간 처리하기위한 최대 클럭 사이클 시간을 Tc라하고, 처리 프레임의 수를 N_{frmae}이라 할 경우 Tc는 식 (6)과 같다.

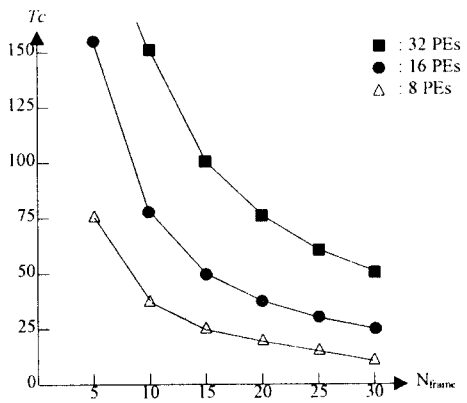
$$Tc \times 1{,}312{,}304 = \frac{1}{N_{frmae}} \text{ sec} \qquad \text{식 (6)}$$

식 (6)을 이용하여 Tc와 프레임 수와의 그래프는 그림 7 (a)와 같다. 그림 7에서 CIF에 대한 연산은 16×16 기준블록과 −8~7 탐색범위를 가정한 것이다. 이 그림에서 QCIF 30 프레임을 실시간 처리를 위해 필요한 Tc는 약 25 nsec이고, CIF 15 프레임일 경우는 약 45 nsec가 된다. 이러한 결과에서 주 클럭 Tc를 약 20 nsec(50 MHz)로 할 경우, 연산에 필요한 데이터의 다운로드 시간을 감안하여도 본 움직임 예측기는 QCIF 30 프레임 또는 CIF 15 프레임을 실시간 처리할 수 있다. 그림 7 (a)에서 얻어진 시간은 16개의 연산기를 이용할 경우에 얻어진 것이다. 그림 7 (b)는 16개의 연산기 처리 시간을 기본 모델로 8개와 32개일 경우를 비교한 것이다. 그림 7 (b)에서 저 전송률 비디오 코덱에 사용되어질 최적의 움직임 예측기의 연산기 갯수가 16개가 됨을 알 수 있다. 즉, 8개의 연산기로 구성된 예측기는 QCIF 30 프레임을 처리하기 위해서는 Tc가 약 12 nsec이며, 이 값은 구현하고자 하는 CMOS 기법으로 동작이 어렵다. 또한, 32개의 연산기로 구성된 예측기는 약 50 nsec이며, 이 값은 연산기가 불필요하게 많이 사용되었음을 알 수 있다. 결과적으로 본 논문에서 제안하고자 한 저 전송률 비디오 코덱을 위한 최적의 움직임 예측기는 클럭 속도

가 약 50 MHz이며, 16개의 연산기로 설계된 선형 세미-시스토릭 어레이 구조가 된다.



(a) QCIF와 CIF의 처리 프레임 수와 Tc



(b) 연산기 갯수에 따른 처리 프레임 수와 Tc

그림 7. 프레임의 종류 또는 연산기의 갯수에 따라 처리되는 프레임 수와 Tc의 관계

Fig. 7 Relation of processing frame counter and Tc with a kind of frame or number of PE

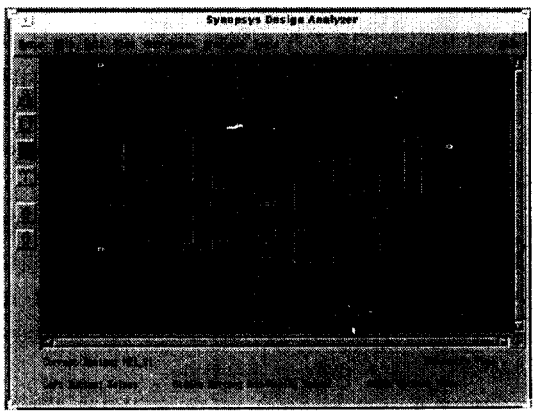
표 3에서는 다른 알고리즘과의 성능 비교가 나타나 있다.

표 3. 선형 세미-시스토릭 어레이 이동예측기 성능분석

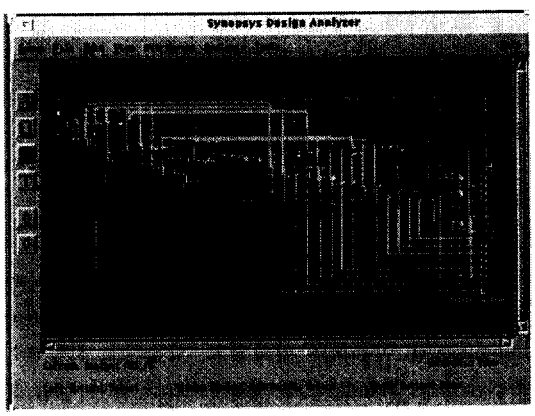
TYPE	Input Pin Count	# of Clock Cycles /Block	# of Clock Cycles /Frame	Internal memory size (bytes)	# of PE
Hsieh & Lin	16	2,209	218,691	1,489	256
Yang	4	8,224	723,798	2,560	32
Ours	3	16,400	1,312,304	1,024	16

표 3에서 클럭 수는 기준 블록이 16×16이고, 탐색 범위가 −16~15 일경우이며, 프레임에 관한 데이터는 입력 비디오 신호가 QCIF에 대한 것이다. 표 3에서 본 논문에서 제안한 이동 예측기는 비교 알고리듬들 중 적은 입력 핀의 수를 가지며, 또한 가장 적은 PE의 갯수를 가진다. 처리 클럭 수를 통하여 PE 효율이 상대적으로 우수함을 알 수 있다. 그리고 내부 메모리의 크기도 다른 알고리듬 보다 적어 ASIC으로 구현할 경우 여러가지 장점을 가진다.

그림 8(a)는 움직임 예측기의 최상위 회로도이고, (b)는 그림 3 (a)의 연산기를 VHDL을 이용하여 설계한 다음 시뮬레이션 과정을 거쳐 마지막으로 논리 합성(Logic-Synthesis)과정을 통해 설계된 도면이다.



(a) 움직임 예측기 논리회로도



(b) PE 논리 회로도

그림 8. 움직임 예측기와 연산기 셀 논리회로도

Fig. 8 Logic Diagram of Motion Estimator and PE

논리 합성 과정은 LG-0.5μ, 3.3V 라이브러리가 이용되었고, 움직임 예측기는 총 17,300 게이트로 구성된다.

또한 본 논문에서 제안하는 움직임 예측기는 H.263 표준에서 움직임 예측기에 요구하는 선택적 추가 기능으로 "Unrestricted Motion Vector mode" 기능이 있다. UMV(Unrestricted Motion Vector) 모드는 가장자리에 위치한 기준블록에 대해 제한된 탐색 범위(예:0~15)가 아닌 일반적인 탐색 범위(예:−16~15)를 적용하는 것이다. 이를 위해 실질적인 탐색영역 데이터가 없는 영역에 가상적인 탐색 영역 데이터를 만들어야 하며, 그 값은 식 (7)과 같다.

$$R_{umv}(x, y) = R_{prev}(x', y'), \quad x' = \begin{cases} 0 & if\ x<0 \\ 175 & if\ x>175 \\ x & otherwise \end{cases}, \quad y' = \begin{cases} 0 & if\ y<0 \\ 143 & if\ y>143 \\ y & otherwise \end{cases} \qquad \text{식 (7)}$$

식 (7)에서 $R_{prev}(x', y')$은 실질적인 이전 프레임의 화소 데이터이고, $R_{umv}(x, y)$는 UMV 모드를 위한 탐색영역 화소 데이터이다. 식 (7)에서 x, y, x', y' 화소 영역의 좌표를 나타낸다. 식 (7)의 구현은 탐색영역 메모리 모듈의 제어부분에 탐색영역의 행 또는 열 형태로 내부 메모리에서 복사가 가능하며, 이 기능은 외부 DMA(Direct Memory Accessor)의 제어부, RISC(Reduce Instruction Set Computer) 프로세서의 제어를 통하여 해결한다. 이러한 내부 메모리내의 복사 기능은 내부 RISC가 복사 기능을 전담하거나, 또는 실질적으로 외부 비데오 메모리에 확장된 영역을 구현하는 기존의 다른 움직임 예측기들에 비교해 제어가 간단하며, 시스템으로 구현할 경우 외부 메모리의 크기를 줄일 수 있는 장점을 가진다.

Ⅶ. 결 론

본 논문의 움직임 예측기는 완전탐색 블록정합 알고리듬을 이용한 선형 세미-시스토릭 어레이 구조다. 또한 저 전송용 비디오 코덱에 사용하기 위한 목적으로 시스템의 최적화를 구현하였다. 최대의 효율을 가지는 연산기와 병렬 구조를 설계하였으며, 연산 결과에 의해 연산기의 갯수를 정하고, 저 전송률 비디오 코덱의 동작 속도를 결정하여 보았다. 결과적으로 제안된 구조는 100%의 연산기 효율을 가지며, VLSI로 구현하기에 적합한 내부 메모리 구조와 최소화는 매우 중요한 요소이다. 본 논문에서는 데이터의 중복 사용을 고려하여 내부 메모리에 최소한의 데이터로 초기화한 다음 연산 수행과 병행하여 연산에 장애를 주지 않으면서 필요한 탐색영역 데이터의 다운로드를 수행 한다. 따라서, 연산기의 효율 증대와 메모리 크기의 최소화, 데이터 입력의 분산으로 시스템의 성능을 향상시켰다.

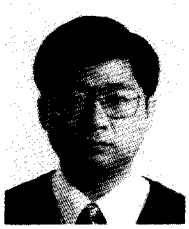
또한 H.263 표준에서 움직임 예측기에 요구하는 선택적 추가 기능으로 UMV 모드 기능을 지원한다.

본 논문의 움직임 예측기는 VHDL 설계와 시뮬레이션 과정을 거쳐, LG-0.5μ, 3.3V 라이브러리를 이용한 논리 합성 과정이 완료되어 총 17,300 게이트로 구성된다. 또한 초당 30 프레임의 QCIF 영상 데이터 처리가 가능한 저 전송률 비디오 코덱의 다른 모듈(DCT, Quantizer, RLC, VLC 등)과 통합되었다.

참 고 문 헌

1. "Video coding for low bitrate communication", Draft ITU-T Recommendation H.263, 1997.
2. F. Thomson Leighton, "Introduction to parallel algorithms and architectures", Morgan Kaufmann Publishers, Inc., '1992.
3. K. M. Yang, M. T. Sun, L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm", IEEE Trans. on circuits ans systems, Vol. 36, No. 10, pp. 1317~1325, Oct. 1989.
4. T. Komarek, P. Pirsch, "Array architectures for block matching algorithms", IEEE Trans. on circuits and system, Vol. 36, No. 10, pp. 1301~1308, Oct. 1989.
5. Hsieh, T. P. Lin, "VLSI architecture for block-matching motion estimation algorithm", IEEE Trans. on circuits and systems for Video Tech., Vol. 2, No. 2, pp. 169~175, June 1992.

6. S. I. Uramoto, A. Takabatake, "A Half-Pel Precision Motion Estimation Processor For NTSC-Resolution Video", IEEE Custom Integrated circuits conference, San Diego, pp. 11.2.1~11.2.4, May 9-12, 1993.
7. J. Baek, S. Nam, M. Lee, C. Oh, and K. Hwang, "A fask array architecture for block matching algorithm", ISCA94, pp. 4.211~4.214, May 30-Jun 2, 1994.
8. Hiroshi Fujiwara, Ming L. Liou, Kun-Min Yang, "An all-ASIC implementation of a low bit-rate video codec", IEEE Trans. on circutes and systems, Vol 2, No. 2, June., 1992.
9. 김기현, 김기철 "선형 시스토릭 어레이를 이용한 완전탐색 블록정합 이동 예측기의 구조", 한국통신학회논문지 제21권 제2호, 1996.



김 기 현(Ki Hyun Kim) 정회원
1989년 : 경북대학교 전자공학과 졸업(학사)
1991년 : 경북대학교 대학원 컴퓨터공학과 졸업(석사)
1991년~현재 : 한국전자통신연구원 선임연구원
※주관심분야 : ASIC 설계, VLSI 구조, 영상처리, 병렬처리

천 대 녕(Dae Nyung Chun) 정회원
1984년 : 경북대학교 통계학과 졸업(학사)
1986년 : 한국과학기술원 전산학과(석사)
1996년 : 한국과학기술원 전산학과(박사)
1996년~현재 : 한국전자통신연구원 선임연구원
※주관심분야 : 영상처리, 컴퓨터비젼, ASIC 설계, VLSI 구조

민 병 기(Byung Ki Min) 정회원
1980년 : 서울대학교 전자공학과 (학사)
1982년 : 한국과학기술원 전기 및 전자공학과(석사)
1991년 : 프랑스 국립 고등 통신원 (ENST) 전자공학과(박사)
1982년~현재 : 한국전자통신연구원 책임연구원
※주관심분야 : 영상처리 알고리듬, VLSI 구조, ASIC 설계