論文95-32B-1-4

CCITT H.261를 위한 효율적인 구조의 움직임 추정 프로세서 VLSI 설계

(An efficient architecture for motion estimation processor satisfying CCITT H.261)

周 洛 炫 *, 金 榮 民 *
(Rackhyun Ju, Youngmin Kim)

요 약

최근 가장 활발한 연구 대상중의 하나인 영상 부호화기 분야에서 움직임 추정 프로세서는 동영상의 부호화에 있어 필수적인 요소로 지적되고 있다. 본 논문에서는 레지스터를 기본 구성 요소로 한 버퍼들 사이의 데이터 흐름을 적절히 조정하는 간단한 구조에 데이터의 병렬처리와 파이프라이닝 기법을 도입함으로써 낮은 화소 전송율(pixel rate)에서도 CIF포맷의 데이터를 최대 30fps까지 처리하여 영상 전화, 영상 회의를 위한 표준 알고리즘인 CCITT H.261 부호화기에 적합한 움직임 추정 프로세서의 구조를 제안하였다.

Abstract

In this paper, we propose an efficient architecture for motion estimation processor which performs one of essential functions in moving picture coding algorithms. Simple control mechanism of data flow in register array which stores pixel data, parallel processing of pixel data and pipelining scheme in arithmetic unit allow this architecture to process a 352×288 pixel image at the frame rate of 30fps, which is compatable with CCITT standard H.261.

I. 서 론

음성, 영상등의 신호처리 방식의 다지틀화추세에 따라 최근 들어 영상신호를 디지틀 방식으로 구현하려는 연구가 활발히 진행되고 있다. 영상전화나 영상회의는 미래 사회에서의 정보 전달 수단중 가장 기본적이면서도 핵심이 되는 부분이므로 영상 부호화 알고리즘의 하드웨어 구현은 최근 선진국 뿐만아니라 국내에서도 가장 활발한 연구가 진행되는 부분중의 하나이다. 이에 따라 1980년대 후반부터 영상 압축 알고리즘의 표준화 작업을 ISO, CCITT등에서 추진해 왔는데 CCITT 추천안 H.261이 최근 영상전화와 영상회의 분야의 표준안으로 채택된 후 이를 하드웨어로 구현하기위한 많은 연구들이 진행되고있다.[1]

특히, 영상 신호를 디지틀 방식으로 표현 했을 경우 저장, 통신, 처리과정등에서 유리한 장점을 가지고 있으나 데이터의 양이 방대하기 때문에 데이터 양을 줄이는 압축 기술이 필수적으로 요구되고 있다. 영상 압축 알고리즘을 하드웨어로 구현하는 방법으로 기존의

* 正會員, 全南大學敎 電子工學科
(Department of Electronics Engineering, Chonnam National University)
接受日字 : 1994年 1月 25日

고성능 DSP를 이용하는 방법이 많이 사용되었으나 최근에는 영상신호를 취급하기위한 전문 프로세서들을 개발하는 추세이며 부호화기내의 특정한 기능을 수행하기위한 프로세서를 이용하여 전체 부호화기를 구성하는 방법도 사용되고 있다. MPEG등의 다른 알고리즘에 비해 처리할 데이터양이 적은편인 CCITT H.261의 알고리즘을 부호화, 복부호화할 수있는 영상 신호처리 전용 프로세서(VSP : Video Signal Processor)는 C-cube사와 IIT사등에서 개발되었으며 알고리즘 수행에 필요한 특정 기능들(DCT, Quantization, Motion Estimation 등)만을 수행하도록 ASIC형태로 설계된 Chip Set들도 INMOS사, LSI-Logic사, SGS-Thomson사등에서 이미 개발되어 상용화된 상태이다.[1] 상용화된 움직임 추정 프로세서의 경우 SGS-Thomson사의 STV3220 프로세서는 CMOS Technology를 이용하여 최대 18Mhz로 화소데이터를 받아들여 8×8, 8×16, 16×16 크기의 매크로 블럭 단위로 초당 30개 CIF프레임의 데이터를 처리하도록 설계되었고 LSI-Logic사에서 개발한 L64720 프로세서도 8×8, 16×16 크기의 매크로 블럭을 단위로 최대 30fps(frame per second)의 속도로 움직임 추정을 수행함으로써 CCITT H.261 부호화기에 사용될 수 있도록 설계되었다.[2][3] CCITT 추천안에서 움직임 추정 및 보상은 옵션으로 규정되어 있으나 연속된 프레임사이에 존재하는 시간 중복성을 제거함으로써 얻어지는 데이터 압축률이 상당하므로 동영상 부호화 과정에서 이를 제외한 부호화기는 생각할 수 없을 것이다. 본 논문에서는 CCITT H.261 동영상 부호화기의 프레임간 예측 부호화에 사용될 수 있도록 CIF포맷의 데이터를 16×16크기의 매크로 블럭 단위로 초당 30 프레임에 대하여 움직임 추정을 수행할 수 있는 움직임 추정 프로세서의 구조를 제안하고 제안된 구조의 성능을 평가하였다.

Ⅱ. 움직임 추정 알고리즘

영상 신호의 압축은 영상신호의 가장 큰 특징의 하나로 지적되는 시공간 중복성을 제거함으로써 이루어진다. 공간 중복성을 제거하는 방법으로는 변환 부호화 방식의 일종인 DCT(Discrete Cosine Transform)를 이용하여 화상이 갖고 있는 에너지를 집중시키는 기법을 CCITT는 물론 MPEG, JPEG등 모든 영상 산호감축 알고리즘이 채택하고 있으며 시간 중복성을 제거는 서론에서 언급한 프레임간의 움직임 예측 보상 기법(interframe motion estimation & compensation)이 주로 사용되고 있다.[4][5] 〈그림 1〉은 CCITT H.261에 따라 구성된 영상신호 부호화기의 블럭도이다. 이 부호화기는 변환부호화와 움직임 보상을 이용한 예측부호화가 결합된 형태로 복합형 부호화기(hybrid encoder)라 불리우고 대부분의 오퍼레이션은 프레임을 일정한 크기로 나눈 매크로 블럭(16×16, 8×8, 등)단위로 수행된다. 〈그림 1〉의 부호화기에서 첫번째 영상은 JPEG과 마찬가지로 블럭 단위의 변환부호화(transform coding)와 양자화(quantization)를 거친후 비 손실형 부호화기(lossless coder)의 일종인 RLC (run length coder), VLC(variable length coder)를 거쳐 BCH CODEC등의 오류 정정 부호화기를 통해 채널로 전송되게 된다.

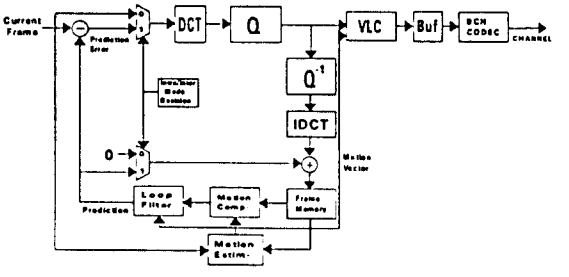


그림 1. CCITT H.261 부호화기
Fig. 1. CCITT H.261 encoder.

반면에 부호화기내의 다른 통로를 따라서 역양자화와 역변환 부호화를 거쳐 재생된 화상은 프레임 메모리에 저장되게 되는데 양자화기 이후의 부호화기는 데이터의 손실이 없는 무손실 부호화기이므로 프레임 메모리에 저장되는 화상 정보는 차후 복호화기에서 재생될 화상과 동일한 데이터를 갖게된다. 이러한 방식으로 첫번째 화상을 부호화한 후 다음 화상부터는 화상간, 즉 이전 프레임의 화상과 현재 입력으로 들어오는 프레임사이의 부호화가 진행된다.[6] 시간 중복성을 제거하는 프레임사이의 예측 부호화는 연속되는 두 프레임내 화상들의 움직임을 탐지하여 이를 프레임 메모리를 통해 보상한 후 보상된 데이터를 현재 프레임의 예측치로 사용하는 기법으로 가능하게되는데 움직임 추정은 이전 프레임내의 화상이 현재 프레임내에서 이동한 정도, 즉 화상 움직임의 크기와 방향을 추출하는 과정을 일컫는다. 움직임을 추정하는 여러가지 알고리즘 중에서 현재 프레임의 한 매크로 블럭과 이에 해당하는 이전 화상내의 매크로 블럭 주위로 일정 크기의 탐

색영역을 정의하고 정의된 탐색영역내부로 현재 프레임의 매크로 블럭을 이동시키면서 가장 잘 정합(matching)되는 부분을 찾아 정합되는 정도와 방향(motion vector)을 계산하는 블럭 정합 방식(BMA : Block Matching Algorithm)이 가장 보편적으로 사용되고 있다.

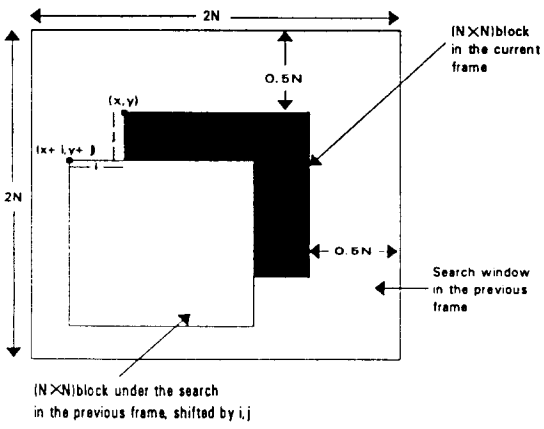


그림 2. 블럭정합 알고리즘
Fig. 2. Block Matching Algorithm.

움직임 추정은 영상 부호화기 내에서 가장 많은 연산이 요구되는 부분으로 가장 빠르게 동작해야할 부분이기도 하다. 따라서 수행과정을 줄임으로써 결과를 빠르게 얻고자하는 여러 가지 탐색기법(CSA, TSS등)들이 제안되었으나 탐색 영역 내부전체를 탐색해가면서 가장 잘 정합되는 부분을 찾는 FSM(Full Search Method, Exhaustive Search)보다는 좋지 못한 성능을 보이고 있고 근래에는 VLSI기술의 발달로 말미암아 FSM를 이용하는 경향이 커져가고 있다.[1], [7], [8] 정합의 정도를 결정하는 조건으로는 CCF(Cross Correlation Function), MSE(Mean Squared Error), MAE(Mean Absolute Error)등이 있으나 일반적으로 곱셈 연산이 필요없고 계산과 구현에 있어 간단한 MAE가 사용되고 있고 실제로 MAE방식은 다른 방법들에 비해 성능면에서도 좋은 특성을 보여주고 있다.[7]

MAE방식의 정합 함수(Matching Function)는

$$Error(i,j) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} |SW(x+i,y+j) - DB(x,y)| \quad (1)$$
$$\left(-\frac{N}{2} \le i \le \frac{N}{2}, \quad -\frac{N}{2} \le j \le \frac{N}{2}\right)$$

$$M = Minimun\ [\ Error(i,j)]$$

Motion vector = *displacement* (i,j) *at which* $Error(i,j)$ *equals to* M
$SW(x,y)$ = *pixel at the position* (i,j) *in search window area*
$DB(x,y)$ = *pixel at the position* (i,j) *in current data block*

로 표현되고 매크로 블럭당 (i,j) 의 가능한 모든 조합에 해당하는 에러를 계산해야한다.

매크로 블럭이 N × N의 크기이고 탐색 영역의 크기가 2N × 2N만큼의 크기로 정해진 경우 각 매크로 블럭마다 $(2N+1)$만큼의 에러가 계산되어야 하고 이들 중 가장 작은 값을 갖는, 즉 현재 프레임의 매크로 블럭과 이와 가장 잘 정합되는 이전 프레임내의 매크로 블럭간의 의 편이 (displacement) (i,j)를 움직임 벡터 (Motion Vector)라하고 움직임 보상시에 가장 중요한 정보로 쓰이게 된다.[8]

Ⅲ. 시스템 구성

영상부호화기를 하드웨어로 구현하는 방법에는 기존의 고성능 마이크로 프로세서를 이용하는 방법과 최근 활발한 진행을 보이고 있는 ASIC을 이용해 실현하는 두가지 방법으로 대별된다. ASIC을 이용하는 방법은 VLSI설계 도구를 이용해 특정기능 만을 수행하도록하여 고속으로 동작하는 프로세서를 구현할 수 있으나 구현후 알고리즘의 추가와 변경이 어렵다는 단점을 안고있다. 그러나 전자의 단점으로 지적되는 낮은 동작속도와 높은 구현 비용을 ASIC을 이용하면 충분히 보상할 수 있기때문에 활발한 진행이 이루어지고 있다.[11] 본 논문에서는 VLSI TEC사의 ASIC설계 Tool인 COMPASS를 이용하여 시스템을 구현하였다. 〈그림 3〉은 제안된 시스템의 전체 블럭도이다. 제안된 구조는 움직임 추정에 필요한 데이터를 저장하는 레지스터 어레이와 레지스터 어레이로부터 입력을 받아들여 식 (1)의 연산을 수행하는 연산부, 에러를 저장하기 위한 RAM, 움직임 벡터를 계산하는 motion vector generator, 전체 시스템의 동작을 제어하는 콘트롤러 등으로 구성되며 16×16크기의 매크로 블럭을 처리할 수 있도록 구성되었다. 한번의 블럭 정합 수행시에는 탐색 영역과 매크로 블럭의 16×16크기의 데이터만 요구되므로 프로세서 내부에 탐색 영역의 모든 데이터를 저장할 필요는 없다. 그리고, 한 매크로 블럭의 우측 16×32 SW(Search Window)데이터는 다음 매크로 블럭의 정합수행시의 좌측 16×32 SW의 데이터로 쓰이게 되므로 프로세서 내부에는 16번의 정합을 수행할 수 있는 16×32크기의 SW데이터와 현재 프레임의 16

×16매크로 블럭 데이터(DB : Data Block)만을 저장한다.

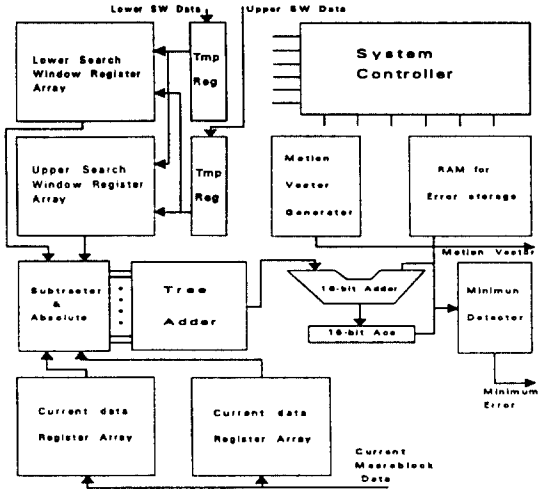


그림 3. 시스템의 블럭도
Fig. 3. Block diagram of proposed system.

〈그림 3〉에서 upper SW 레지스터 어레이와 lower SW 레지스터 어레이는 수직 방향으로 블럭 정합 수행시 탐색 영역내의 16×32 데이터중 각각 윗 부분의 16×16영역에 해당하는 데이터와 아랫 부분의 16×16 영역에 해당하는 데이터를 저장하는 부분이다.

〈그림 4〉는 FSM방식으로 블럭 정합을 수행할 때 좌측 상단부를 확대한 그림이다.

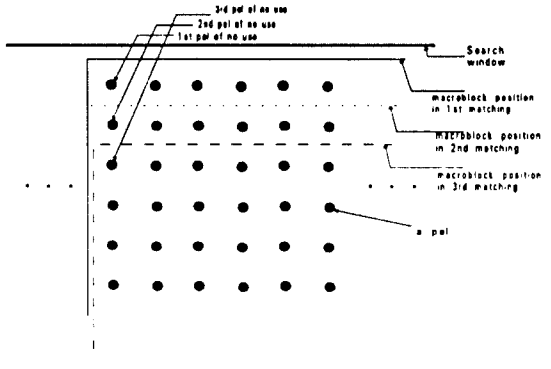


그림 4. 정합 수행 과정
Fig. 4. Block matching process.

그림에서 알 수 있는 바와 같이 수직방향으로 한번의 정합이 수행될 때마다 차후의 정합에는 쓰이지 않는 화소데이터가 한개씩 좌측 상단에서부터 수직 방향으로 발생하며 16번의 정합이 완료되면 좌측 32개의 화소 데이터가 필요없는 데이터로 남게된다.

본 구조는 이를 이용하여 매 정합마다 필요없게 되는 화소 데이터를 대신하여 다음 정합에 필요한 새로운 화소 데이터(32개)를 받아들여 순차적으로 이를 저장하며 16번의 정합후 입력되어있을 16개의 새로운 데이터는 좌측으로 화소 단위의 쉬프트 오퍼레이션이 수행될 때 새로운 위치로 이동되며 쉬프트 오퍼레이션에 의해 새롭게 저장된 16×32크기의 데이터를 이용하여 다음 16번의 정합이 수행된다.

〈그림 5〉는 설계된 SW 레지스터 어레이와 입력 데이터를 임시로 저장하는 temporary 레지스터의 블럭도이다.

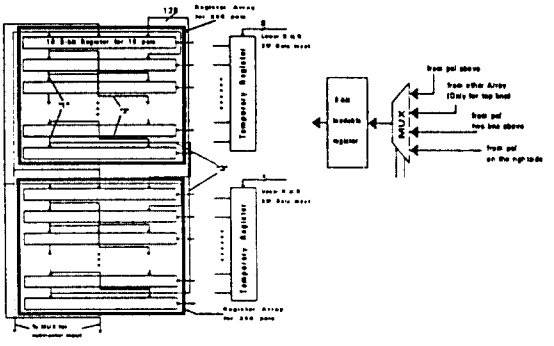


그림 5. SW 레지스터 어레이와 화소 저장 레지스터의 블럭도
Fig. 5. Block diagram of SW register array & pixel storage register.

SW 레지스터 어레이의 데이터중에서 연산부의 입력으로 작용하여 직접 작용하는 데이터는 upper SW 또는 lower SW의 최하단 레지스터의 출력인데 연산에 필요한 데이터를 적절하게 출력시키기 위해 각 라인 레지스터(16×1의 화소를 저장)의 데이터가 "1"(인접 라인 레지스터간의 데이터 이동), "2"(한 라인 건너 위치한 라인 레지스터간의 데이터 이동)의 통로를 통하여 이루어질 수 있도록 하였으며 레지스터 어레이의 최상층에 위치한 라인 레지스터의 경우는 다른 레지스터 어레이의 하단 두번째 라인 레지스터의 출력("3")도 입력으로 연결될 수 있도록 설계되었다. 레지스터 어레이의 내부 데이터는 16×1크기의 라인 단위로 연산부에 입력될 수 있도록 128bit단위로 이동되며 128bit의 데이터는 16개의 화소 데이터에 해당된다. 라인 레지스터 내부에는 화소를 저장하는 16개의 8bit

레지스터가 있고 이 화소 저장 레지스터는 〈그림 5〉에서 나타난 바와 같이 위에서 기술된 입력외에도 우측에 위치한 8bit화소 데이터가 MUX를 통하여 추가로 입력될 수 있도록 구성되어 있다.

설계된 SW 레지스터 내부에서 데이터들의 이동은 한 블럭 정합시 필요한 데이터 순환과 한 블럭 정합완료후 다음 정합을 실행하기 위한 데이터들의 이동, 수직방향으로 16번의 정합 후 다음 16번의 정합을 수행하기위한 화소단위 쉬프트 오퍼레이션의 3종류로 구분되며 모든 데이터들의 이동은 클럭 단위로 이루어진다. 이들 데이터의 순환은 각 화소 저장용 레지스터의 입력측에 연결되어 있는 MUX의 제어신호들에 의해 직접 조정되는데 이 신호들은 콘트롤러가 시스템 클럭과 외부 데이터의 입력을 알리는 신호(NEXT)의 수를 세어 발생시킨다.

두개의 temporary레지스터는 위에서 설명된 새로운 화소 데이터를 받아들여 이를 순차적으로 저장하고 있다가 쉬프트 오퍼레이션이 이루어질때 SW레지스터 어레이의 우측 32개의 화소 저장 레지스터로 데이터를 전달하는 역활을 수행한다. 〈그림 5〉에 나타난 블럭내부의 전체적인 데이터 흐름은 temporary 레지스터를 통해 새로운 데이터가 입력되는 동시에 upper SW또는 lower SW레지스터 어레이의 최 하단 레지스터를 통해 연산에 직접 참여하는 데이터가 출력되는 것으로 요약된다. 레지스터 어레이로부터의 출력과 함께 연산부의 입력으로 작용하는 현재 프레임의 매크로 블럭(DB : Data Block) 데이터는 〈그림 3〉의 Current data register array에서 출력된다. 데이터를 저장하는 DB 레지스터 어레이는 16×16크기의 화소를 저장할수 있고 라인단위(128bits)로 데이타를 circulation 시킬수 있도록 구성된 동일한 두개의 모듈로 구성되어 있다. 두개의 동일한 모듈중 한개만 직접 연산에 참여하게 되고 다른 한개는 연산이 다른 모듈에 의해 연산이 수행되는 동안 inactive한 상태로 다음번 매크로 블럭의 데이터를 받아들여 저장한다. Active 모듈에 저장된 블럭의 정합이 완료된 후에는 이미 새로운 매크로 블럭의 데이터가 inactive모듈에 저장되어 있게 되고 두 모듈의 역활이 서로 바뀌게 되어 inactive모듈에 저장되어 있는 데이터가 실제로 연산에 참여하게 되고 active모듈은 다음번 정합에 사용될 새로움 매크로 블럭 데이터를 입력으로 받아들여 저장하게 된다. SW 레지스터 어레이에서의 두 128bit출력과 DB레지스터 어레이에서의 두 128bit출력은 각기 Mux를 통하여 연산부의 시작부분, 즉 감산기의 입력으로 작용하여 연산에 직접 쓰이게 된다.

〈그림 6〉은 식(1)의 연산을 수행하는 연산부의 블럭도를 보여주고 있다. 연산부의 구성은 감산기와 절대값을 구하는 회로, tree adder, adder-accmulator회로 comparator등으로 이루어진다. 감산기와 감산결과의 절대치를 구하는 회로는 식(1)의 연산을 수행하기 위한 기본 요소이며 tree adder는 감산기와 절대값을 구하는 회로를 거쳐 계산된 16개의 8bit 화소 정합 에러를 합산하기 위하여 설치되었다.

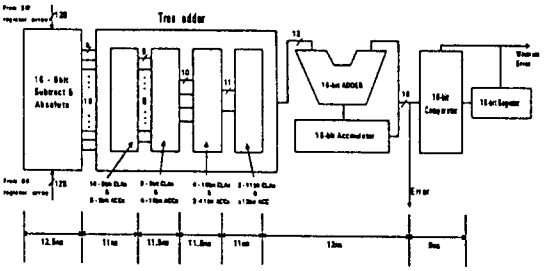


그림 6. 연산부의 블럭도와 타이밍
Fig. 6. Block diagram & timing specification of arithmetic unit.

각 단(stage)에서의 가산기는 n-bit의 입력을 받아들여 (n+1)bit의 결과를 출력하도록 설계되어 계산과정에서 overflow가 발생할 수 없도록 처리하였으며 tree adder 뿐만 아니라 연산부 내의 모든 가산기는 CLA(Carry Lookahead Adder)를 사용하였다. tree adder를 거쳐 계산된 결과는 12bit로 출력되며 이는 한 라인의 정합 에러에 해당하는 값이므로 16개의 라인 정합 결과를 합산하기 위하여 16-bit add-accmulate구조를 도입하였다. 16bit는 블럭 정합 결과로 발생할 수 있는 최대 에러를 저장하기에 충분한 크기이다. 16번의 라인 정합을 수행한후에 accmulator에는 주어진 편이(displacement)에 대한 블럭 정합 에러가 저장되어있게 되고 이 정합 에러는 comparator의 입력과 에러를 저장하기위한 RAM의 데이터 버스로 입력된다. Comparator는 새로 계산되어 입력된 데이터와 이전의 다른 편이에서의 에러값을 저장하고 있는 레지스터의 출력을 비교하고 작은 에러값은 차후에 계산될 다른 에러값과의 비교를 위해 레지스터에 저장 된다. 한 매크로 블럭의 정합이 완료되었을 때 이곳에 저장된 데이터는 주어진 SW 내에서의 최소 블럭 정합 에러가 되고 정합이 완료되었음을 알리는 신호후에 출력단자로 출력된다. 〈그림 3〉에서는 comparator와 최소 에러를 저장하기 위한 16bit레지스터를 합하여 최소 에러 검출기(minimum detector)블럭으로 표기되어있다. 〈그림 6〉에는 연산부의

블럭도와 함께 각 단(stage)마다 최대로 소요되는 시간이 표시되어있다. 연산부의 각 단(stage) 사이에는 레지스터를 둠으로써 요구되는 동작시간에 적합하도록 파이프라인 기법을 이용하여 처리하였다.

초당 30fps의 의 처리속도와 CIF데이터를 기준으로 하여 한 프레임을 16×16의 매크로 블럭으로 나눈다면 한 프레임내에는 22×18 개의 매크로 블럭이 존재하게 된다.

30fps의 처리속도를 얻기위해 한 매크로 블럭의 정합에 허용되는 시간은 약 85μs로 제한을 받게 된다. 주어진 탐색영역내(32×32)에서 FSM방식으로 한개의 매크로 블럭의 정합이 끝나기까지는 172번의 정합이 요구되는데 현재 프레임내의 한 매크로 블럭과 탐색영역내의 16×16크기의 블럭과의 정합에 주어지는 시간은 290ns로 제한된다. 290ns의 주어진 시간내에 식(1)의 연산을 수행하기위하여 본 구조에서는 16개의 화소 데이터 즉 매크로 블럭의 라인단위로 연산을 수행한 후 이를 합산하는 구조를 채택하였다. 한 라인의 정합 에러를 계산하는데는 약 18ns의 시간이 허용되는데 한번에 라인 정합 에러를 구하기에는 너무도 작은 시간이므로 파이프라이닝 기법의 도입이 필수적으로 요구된다.

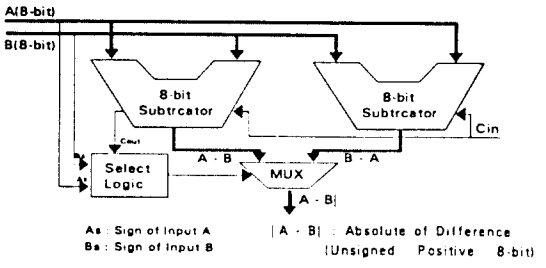


그림 7. 차의 절대값을 구하기 위한 회로.
Fig. 7. Subtract & Absolute Logic.

〈그림 7〉은 〈그림 6〉에서의 Subtract & Absolute Logic을 보여준다. 두개의 감산기를 병렬로 연결하여 MUX를 통해 한개의 8-bit(Positive) 결과를 출력시킴으로써 계산 지연 시간을 줄일 수 있었다. 두개의 감산기는 각각 CLA를 이용해 구성되었고 감산기 내부의 최대 지연 시간은 6개의 게이트를 통과하는 시간이다. Treeadder 내부의 9-bit 이상의 가산기는 최소의 게이트를 통과하도록 설계된 두개의 sub-CLA를 직렬로 연결하여 구성하였으며 11-bit 가산기의 경우 최대 9개의 게이트를 통과하는 시간이 최대 지연 시간이 된다. 연산부 내부의 계산에서 최대 bottle neck은 라인 정합 에러를 합산하기위한 16-bit 가산기이며 계산 시간을 줄이기위해 본 논문에서는 COMPASS에서 제공되는 최대 지연 시간이 10.3ns인 CSA(Carry Select Adder)를 채택하였다. 비교기 역시 두 입력을 병렬로 비교하여 최대 9개의 게이트를 거친수 결과가 계산되도록 설계되었다. 참고로, 본 논문의 시뮬레이션은 0.8 μm CMOS Technology를 이용하여 수행되었다(예: 4개의 게이트를 구동하는 2-input NAND게이트의 지연시간은 약 0.7ns).

설계된 연산부는 〈그림 6〉에서 보여지는 바와 같이 파이프라인 레지스터의 출력에서부터 각 단에서의 최종 결과가 산출되는데 걸리는 지연시간은 18ns보다 충분히 작게 설계되어 있으므로 연산부의 처리능력은 매초당 30개 CIF프레임 이상이라고 할 수 있다.

레지스터 어레이와 연산부를 거쳐 계산된 각 블럭 정합 에러들은 comparator의 입력 뿐만 아니라 289×16bit 크기의 블럭 정합 에러를 저장하기 위한 RAM에 저장되게된다. 에러 값들을 굳이 저장해 놓을 필요는 없지만 정합 결과를 알고자 할 필요가 있을 때 외부에서 address를 발생시켜 해당하는 편이에서의 정합에러를 확인하도록 하기위하여 에러를 저장하는 RAM을 두었다. Motion vector generator에서는 움직임 벡터를 계산하는 역활을 수행하는 부분으로 각 편이에 해당하는 블럭 정합 에러가 계산될 때마다 이에 대응되는 벡터를 계산하는데 이 벡터는 에러값이 RAM에 쓰여질 때 쓰여질 번지를 계산하는 정보로 이용되게 된다. Motion vector generator내부에는 최소 에러 검출기 내부의 레지스터에 저장된 에러값을 갖도록 하는 편이(displacement)를 저장하는 레지스터가 있어 최소 에러 값을 갖는 편이를 추적하는데 한 매트로 블럭의 정합이 완전히 끝났을 때 이곳에 저장된 편이는 움직임 벡터가 되며 정합수행이 완료되었음을 알리는 신호 후에 출력단자로 출력되게 된다. 전체 시스템을 조정하는 콘트롤러는 주로 시스템 클럭과 매크로 블럭과 탐색영역 데이터 입력을 알리는 외부 입력 신호의 수를 세는 카운터로 구성되어 있고 카운트된 숫자에 따라 레지스터 어레이내의 데이터 교환을 조정하는데 필요한 MUX 콘트롤 신호들, 연산부 레지스터의 데이터 저장여부를 조정하는 신호, 움직임 벡터를 계산하는데 필요한 신호, RAM 콘트롤 신호등을 발생시킨다.

이상에서 〈그림 3〉의 구조로 제안된 시스템의 동작 특성과 시스템 내부에서의 데이터 흐름을 기술 하였다. 개괄적으로, 본 구조는 정합시 마다 차후 연산에 사용되지 않는 1개의 화소 데이터가 발생한다는 점에 착안하여 이를 대치할 새로운 데이터 입력을 받아들여 저

장하였다가 레지스터 어레이내의 데이터와 함께 순환, 이동시키는 간단한 데이터 제어 방식과 연산부의 파이프라이닝 기법으로 CCITT H.261을 만족하는 동작속도를 갖도록 구성되었다.

Ⅳ.성능 평가 및 분석

FSM방식을 이용한 움직임 추정은 방대한 계산이 요구된다는 점을 제외하면 탐색후 결과에 대한 신뢰도가 가장 크고 탐색과정에서의 규칙성때문에 타 알고리즘보다 쉽게 VLSI로 구현될 수 있다는 특성을 가지고있다. 따라서, 1980년대 중반까지의 연구는 주로 움직임 추정시 탐색점(Search Point)를 줄이는 방법으로 빠른 결과를 얻고자 하는 알고리즘에 관한 연구가 주를 이루었던 반면, 급속한 발전을 보이고 있는 VLSI기술에 의해 최근의 연구와 시제품들은 FSM방식을 이용하는 경향이 커지고있다. FSM방식을 이용한 프로세서들의 구조들은 Systolic Array형태를 이용하는 방법과 내부에 화소를 저장하는 장소를 두고 이 곳으로부터 데이터를 병렬로 엑세스한 후 주연산부에 파이프라인 기법을 도입해 이들 데이터를 병렬로 처리하는 방법으로 대별된다.[8][9][10] 물론, 전자의 경우도 어레이 내부의 PE(Processing Element)에 파이프라인 기법을 사용하며 주로 레지스터를 이용해 PE간에 데이터들을 전달한다.[9][10] 본 논문에서는 후자의 방법을 따라 화소데이터를 시스템 내부의 레지스터 어레이에 저장한 후 16개의 화소 데이터를 연산부에서 동시에 처리하는 기법을 이용하였다. 〈표 1〉에서는 최근 연구된 결과와 본 논문에서 제안된 시스템의 성능을 비교하여 보여주고있다.

표 1. 제안된 시스템의 성능 비교
Table 1. Comparison of motion estimator.

구분 / 비교대상	L64720	STV 3220	REF[8]	REF[9]	REF[10]	Proposed system
Max Frame rate	30Hz	30Hz	20Hz	15Hz	11Hz	30Hz
Data Format	CIF	CIF	256×240	CIF	CIF	CIF
Block size	16×16 or 8×8	16×16 8×8 8×	16×16	16×16	8×8	16×16
Pixel storage	RAM	RAM	RAM	Register	RAM	Register
Control Overhead	High	High	High	Medium	Medium	Medium
Expansion Overhead	High	High	High	Medium	Medium	Medium
Max Clock	30/40MHz	3/12MHz	20MHz	25MHz	40MHz	56MHz

CCITT H.261에서 규정한 영상전화와 영상회의 시스템은 입력 화상의 크기가 352×288인 CIF화상데이터를 화상율(Frame Rate)은 7.5Hz이상으로 처리하도록 권고되어있다. 7.5Hz의 낮은 화상률은 동화상중 움직임이 적은 영상전화나 영상회의 시스템 적용을 고려한 경우지만 MPEG등의 고속 알고리즘에서는 30Hz 이상의 고속 동작을 요구하고있다. Systolic array형태로 구현된 프로세서들의 경우 PE의 숫자를 늘림으로써 빠른 연산속도를 얻을수는 있지만 VLSI구현시 PE갯수에 따라 면적이 증가할 수 밖에 없다.

따라서 이 경우 PE를 간략화하는 것이 빠른 동작속도를 얻을 수 있는 가장 좋은 방법이지만 식(1)의 연산을 수행하기 위해서는 기본적으로 감산기와 절대값을 구하는 회로, 가산기, Accmulator등이 필수적으로 PE에 포함되어야하며 데이터의 흐름을 제어하기위해 중간값을 저장하고 순환시키는 레지스터들이 부가되어야만 한다.[9,10] 그러므로, PE를 줄이는 데는 한계가 있을 수 밖에 없고 대부분의 경우 데이터들이 내부적으로는 레지스터를 통해 전달되기때문에 적용분야의 속도에 맞추어 PE의 갯수를 늘림으로써 적절한 속도로 동작 스피드를 향상시킬 수 있다는 장점이 있다.[10] 그러나 이 경우도 내부에 설치되어 있는 데이터 저장 장소가 작기때문에 데이터를 외부(Frame Memory)에서 중복시켜 입력시켜야하는 제어 부담을 안게 된다.[9,10] 본 논문에서는 Systolic Array기법의 레지스터간 데이터 이동을 레지스터 어레이로 집중시켜 간단한 제어를 통해 연산부로 16개의 화소 데이터씩 출력되도록 구성하였다. 반면, 최근의 연구 결과물이나 시제품들은 내부에 화소 데이터를 저장하는 장소를 두고 이곳으로부터 데이터를 엑세스하여 처리하는 기법을 주로 이용하고 있으며 대부분 데이터 저장 장소로 메모리(SRAM)을 이용한다.[2][3][8] 이 경우, 주 연산부의 파이프라인 수를 늘림으로써 높은 동작 속도를 확보할 수 있으나 처리할 데이터를 메모리에서부터 엑세스해야하므로 데이터 엑세스 시간이 파이프라인수를 증가시키는 데 있어 가장 큰 제한 요인이 된다. 따라서, 빠른 엑세스 시간을 갖는 RAM의 개발 없이는 특정 속도 이상의 동작이 불가능하며 RAM의 사용에는 필연적으로 어드레스 계산의 전제가 있어야 하므로 이를 계산하기위한 부분을 추가해야하며 전체 시스템에 대한 제어부담이 늘게된다.[8]

본 논문에서는 화소데이터를 저장하는 장소로 RAM을 사용하지 않고 레지스터를 사용함으로써 엑세스시간에 관계없이 시스템의 동작 속도를 향상시킬수 있도록 설계하였다. 일반적으로 레지스터는 RAM보다 많은 면적을 차지하므로 본 논문에서는 탐색영역의 모든 데이터를 저장하지 않고 16×32개의 데이터만을 저장하고 이후의 데이터는 외부로부터 serial하게 받아들여

라인 버퍼에 저장 하였다가 이를 병렬로 어레이내부로 이동시킴으로써 입력 핀수를 줄이고 약 3MHz의 낮은 대역폭으로 외부 프레임 메모리(일반적으로 DRAM)를 엑세스 하며 데이터를 중복해서 입력해야하는 부담을 제거함으로써 데이터 입출력에 관한 제어부담을 줄일 수 있었다. 연산부의 설계는 식(1)의 연산을 수행하기 위한 기본 요소들(감산기, 가산기, 비교기, Accmulator, 등)을 Tree구조 형태로 배열하여 한 곳에 집중시킴으로써 파이프라인 기법이 쉽게 적용될 수 있고 차후의 더욱 복잡하고 고속동작을 요구하는 알고리즘에 적용될 수 있도록 하였다. 현재 본 논문에서 제안된 구조는 는 CIF데이터를 처리할 수 있도록 레지스터 어레이를 구성하였으며 연산부는 30fps까지의 동작이 가능하도록 설계되었으나 연산부의 파이프라인 수를 늘이는 등의 간단한 수정을 통해 고수준의 알고리즘에 쉽게 적용될 수 있도록 확장될 수 있다. 본 논문에서 제안한 구조의 각 모듈과 전체 시스템의 올바른 동작을 통해 움직임 벡터와 최소 에러값이 정확히 산출됨을 확인하였다. 본 구조는 16×16 크기의 매크로 블럭을 기준블럭으로 -8/+7 범위를 대상으로 정합을 수행하며 매크로블럭의 정합을 완료하는데 걸리는 클럭수는 파이프라인 latency 사이클을 포함하여 5104 사이클이며 이는 CIF 입력 데이터의 경우 30fps의 동작 속도를 확보할 수 있는 속도이다.

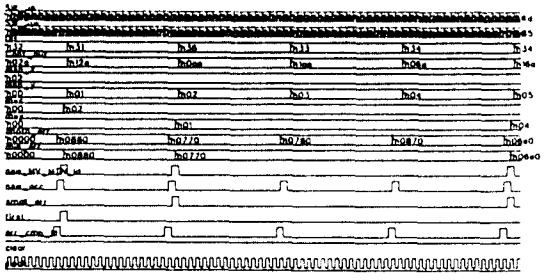


그림 8. 정합 수행시 시뮬레이션 결과
Fig. 8. Simulation result in block matching process.

〈그림 8〉은 50Mhz의 동작 주파수로 시뮬레이션을 수행한 결과이다.

V. 결 론

이상에서 CIF포맷의 데이터를 최대 30fps로 처리할 수 있는 움직임 추정 프로세서의 구조를 제안하고 성능을 평가하였다. 설계된 프로세서는 0.8μm CMOS technology를 이용하여 검증되었고 약 137,300개의 게이트로 구성된다. 본 구조는 최대 56Mhz의 주파수로 동작할 수 있으며 화소 데이터가 입력되는 pixel rate는 306ns까지 가능하다. 매크로 블럭의 데이터와 탐색 영역의 화소데이터를 16개 즉, 라인 단위로 처리하고 연산부에는 파이프라이닝기법을 도입함으로써 306ns의 낮은 pixel rate에서도 CCITT H.261을 만족시키는 동작 속도를 얻을 수 있었다. 본 논문에서 제안된 구조에서는 데이터를 레지스터 어레이에 저장함으로써 메모리에 데이터를 저장했을 경우 고속 동작 실현에 있어 최대 문제점인 엑세스 시간에 의한 문제를 제거하여 연산부의 파이프라인 숫자를 늘림으로써 시스템의 스피드를 향상시킬수 있었고 어드레스를 제어할 필요없이 어레이 내부의 MUX들을 제어함으로써 필요한 데이터를 간단히 제어할 수 있었다. 또한 외부 메모리(Frame memory : 주로 DRAM)에서 데이터를 중복시켜 입력시킬 필요가 없으므로 전체 부호화기 시스템의 구성도 간단하게 이루어질 수 있다. 이상 위에서 열거된 이유들로 인해 본 논문에서 제안된 구조는 시스템의 확장이 간단하게 이루어질 수 있고 다른 고속 알고리즘에도 손쉽게 적용될 수 있다. 이후의 연구는 16×16의 크기를 갖는 매크로 블럭외에 여러가지 크기의 매크로 블럭을 처리할 수 있도록 시스템을 확장하고 MPEG등의 상위 수준의 표준안에도 적용될 수 있도록 시스템의 속도를 향상시키는 방향으로 수행되어야 한다.

참 고 문 헌

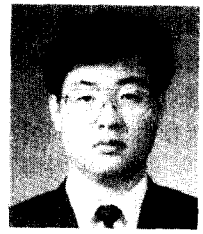
[1] 최상훈외 4명 "재구성 가능한 다중 프로세서 시스템을 이용한 혼합 영상 부호화기에 관한 연구" 전자공학회 논문집 Vol.30-B No.10 pp. 1-4
[2] LSI-Logic Corp. "L64720 Video Motion Estimation Processor".
[3] SGS-Thomson Corp "STV3220 Motion Estimation Processor".
[4] Majid Rabbani, Paul W. Johns "Digital Image Compression Technique" SPIE Optical Engineering Press pp. 79-83, 108-111.
[5] 김한수 "영상 전화 시스템에서의 신호처리 기술" 자공학회지 Vol.20 No.10 pp.7-13
[6] LSI Logic Corp "VIDEO COMPRESSION

CHIP SET".
[7] M. Ghanbari "The Cross-Search Algorithms for Motion Estimation" IEEE Trans on COMM Vol.38 No.7 1990 pp.950-953.
[8] Yeu-Shen Jeung, Liang-Gee Chen "An Efficient and Simple VLSI Tree Architecture for Motion Estimation Algorithms" IEEE Trans on Signal ProcessingVol.4 No.2 1993 pp.889-898.
[9] Kun-Min Yang, Ming-Ting Sun, Lancelot Wu "A Family of VLSI Designs for the Motion Compensation Block-Matching Algorithm" IEEE Trans on Circuit & Systems Vol.36 No.10 1989 pp.1317-1325.
[10] Thomas Komarek, Peter Pirsch "Array Architecture for Block Matching Algorithms" IEEE Trans on Circuit & Systems Vol.36 No.10 1989 pp.1301-1308.

저 자 소 개

金 榮 民(正會員)

1954년 4월 18일생. 1976년 2월 서울대학교 전자공학과 졸업 (공학사). 1978년 2월 한국과학원 전기및전자공학과 졸업 (공학석사). 1978년 3월 ~ 1979년 7월 한국선박해양 연구소 (주임연구원). 1979년 8월 ~ 1982년 7월 국방과학연구소 (연구원). 1986년 오하이오 주립대학교 전기공학과 (공학박사). 1988년 6월 ~ 1991년 8월 한국전자통신연구소 (실장). 1991년 9월 ~ 현재 전남대학교 전자공학과 교수. 주요 관심분야는 영상압축, VLSI 설계, 신경회로망 등임.



周 洛 炫(正會員)

1970년 10월 27일생. 1993년 2월 전남대학교 전자공학과 (공학사). 1993년 3월 ~ 현재 전남대학교 전자공학과 대학원 재학중. 주요 관심분야는 VLSI 설계, 영상압축 등임.