고속 움직임 예측기 구현에 관한 연구

정회원 김 진 연*, 박 상 봉***, 진 현 준**, 박 노 경*

A Study on Implementation of the Fast Motion Estimation

Jin-yean Kim*, Sang-bong Park***, Hyun-jun Jin**, Nho-kyung Park* ***Regular Members***

요 약

오늘날 통신 기술이 나날이 발전하고 있지만 디지털 영상신호가 방대한 데이터를 가지고 있기 때문에 데이터의 저장, 처리 및 전송을 위해서는 보다 많은 데이터 압축이 필요하게 되었다. 이에 따라 ITU-T에서는 디지털 영상신호의 압축 표준을 위해서 H.26x 등을 제정하였다. 일반적으로 영상처리에서는 픽쳐간 상관 관계를 이용하여, 픽쳐간의 움직임 예측을 통한 시간적 중복성을 제거하여 데이터를 크게 압축하는 것이 많이 사용되고 있다. 대부분의 비디오 코딩 시스템에서 움직임 예측/보상(Motion Estimation/Compensation)방법으로 블록 정합 알고리즘을 사용하는데 이는 특정한 비용 함수의 최소 값을 기반으로 사용되고 있다. 그러나 이 방법은 많은 수의 계산을 필요로 하여 탐색 시간이 오래 걸리는 단점이 있다. 따라서 H.26x에서와 같은 실시간 저비트율 부호화를 위해서는 전역 탐색법 보다는 효율적인 고속 탐색 알고리즘이 효과적이다.

본 논문에서는 움직임 예측에 소요되는 탐색 시간을 줄이기 위해서 고속 탐색 알고리즘 중에서 Nearest-Neighbors 탐색 알고리즘을 이용하여 움직임 예측기를 FPGA로 설계하였으며, VHDL로 코딩(Coding)하고, Xilinx Foundation을 이용하여 설계 및 검증하였다.

ABSTRACT

Since digital signal processing for motion pictures requires huge amount of data computation to store, manipulate and transmit, more effective data compression is necessary. Therefore, the ITU-T recommended H.26x as data compression standards for digital motion pictures. The data compression method that eliminates time redundancies by motion estimation using relationship between picture frames has been widely used. Most video conding systems employ block matchig algorithm for the motion estimation and compensation, and the algorithm is based on the minimun value of cost functions. Therefore, fast search algorithm rather than full search algorithm is more effective in real time low data rates encodings such as H.26x.

In this paper, motion estimation employing the Nearest-Neighbors algorithm is designed to reduce search time using FPGA, coded in VHDL, and simulated and verified using Xilink Foundation.

Ⅰ. 서 론

서로 다른 두 개의 영상 집합사이에 비슷한 정도를 측정하는 영상 정합은 패턴 인식, 컴퓨터 비젼, 영상 처리에 있어서 아주 중요한 문제로 대두되어 왔고 여러 가지 영상 정합 방법이 제안되어 왔다. 이 연산들은 특징 검출, 거리 변화나, 정합도를 측정하여 영상의 유사도를 측정하게 된다. 이러한 측정이 움직임 예측에도 사용되고 있다. 영상처리에서는 화상간 상관관계를 이용하여, 화상간의 움직임 예측을 통한 시간적 중복성을 제거하여 많은 데이터를 압축한다.

대부분의 비디오 코딩 시스템에서는 움직임 예측/보상(motion estimation/compensation)을 하기 위해,

* 호서대학교 정보통신공학과 회로 및 시스템 설계 연구실(esdamars@asic.hoseo.ac.kr),
** 호서대학교 정보통신공학과 응용소프트웨어 연구실(*hjjin@office.hoseo.ac.kr*), *** 세명대학교 정보통신학과 *IT ASIC* 연구실
논문번호 : K01154-0709, 접수일자 : 2001년 7월 9일

영상을 정합하는 알고리즘으로 제안된 것이 여러 가지가 있다. 그중에 화소 순환 알고리즘(pel recursive algorithm), frequency-domain 알고리즘, gradient 알고리즘, 블록 정합 알고리즘(block matching algorithm)들이 쓰이고 있다.[1]

이 중에 블록 정합 알고리즘은 현재 프레임의 블록 움직임을 과거 프레임의 탐색 영역에서 예측하는데 일반적으로 많이 사용되고 있다. 그러나 블록 정합 알고리즘을 이용할 때 필요한 많은 계산량에 의해 이의 실시간 처리에 어려움이 있다. ITU (International telecommunication Union)에서는 영상 소스 코딩 규격으로 H.263을 제안하였는데, 이 규격에서는 계산량을 줄이기 위한 해결책으로 블록 정합을 고속으로 탐색하는 알고리즘으로 Nearest Neighbors 탐색법이 제안되었다.[3],[4]

ITU-T에서 제정한 표준안인 H.263은 64kbps이하의 낮은 비트율에서의 시청각 동영상 서비스지원을 목적으로 한다. 이와 같은 저 비트율에서의 실시간 부호화를 위해서는 고속으로 블록정합을 탐색하는 고속탐색법이 중요한 요소이다. 최근에 제한된 nearest-neighbors 탐색법은 화상전화 등을 대상으로 하는 저비트율 비디오 코딩에 매우 효율적인 방법으로 H.263의 부호화 권고안인 TMN8(Test Models for the Near-Term, Version8)에 고속 탐색법의 방안으로서 제시되기도 하였다.

Nearest-Neighbors 탐색 알고리즘은 1997년 ITU-T에서 제안된 방식으로 Image-processing 관련 분야에서 S/W로 구현 기법을 분석하고 있는 실정이다.

본 논문은 H.263과 같은 동영상 코덱에 적용할 수 있는 움직임 예측기를 하드웨어로 설계하기 위해서, Nearest Neighbors 탐색 알고리즘을 이용한 고속 움직임 예측기를 FPGA로 설계하였다. 또한 설계한 움직임 예측기의 기능과 타이밍 시뮬레이션을 위해 VHDL로 코딩하고 Xilinx Foundation을 이용하여 설계 및 검증하였다.

본 논문은 전체 5장으로 구성되어 있고, 2장에서는 H.263에 사용되는 Nearest Neighbors 탐색법을 이용한 움직임 예측에 대해 알아보고, 3장에서는 이를 하드웨어로 구현한 것을 보여주며, 4장에서는 설계한 하드웨어를 모의 실험을 통한 검증을 하였다. 마지막으로 5장에서는 설계한 고속 움직임 예측기에 대해 간단히 정리하여 결론을 맺었다.

Ⅱ. 움직임 예측

2.1 부호기 개요

H.263의 기본적인 영상 소스 코딩 알고리즘은 H.261에 기반하고 있다. 그러나 H.261에 비하여 같은 화질의 영상에 대해 거의 반정도의 비트를 생성한다. 이러한 저비트율 비디오 코딩은 움직임 예측 알고리즘을 사용함으로써 가능한 것이다. 움직임 예측은 대개 하드웨어 구현이 쉬운 편인 전역 탐색 알고리즘을 이용하였다. 그러나 전역 탐색 알고리즘은 탐색창 전부에 대한 계산을 함으로써 많은 계산량이 문제가 되었다. 이러한 문제를 해결하기 위해 많은 고속 탐색 알고리즘이 제안되었다. 그 중 ITU-T에서 권장한 Nearest Neighbors 탐색 알고리즘을 본 논문에서 VHDL로 구현하고, 설계하였다.

본 논문에서는 H.263 version2의 시험 모델인 TMN(Test Models for the Near-term)8을 중심으로 구현하였다. 그림 1은 H.263 엔코더 블록으로 그림에서 선택된 블록은 움직임 예측을 하는 부분을 보여주고 있다. 본 논문에서는 이 움직임 예측 블록을 FPGA로 설계하고, VHDL로 코딩하여 Xilinx Foundation을 이용하여 구현하였다.

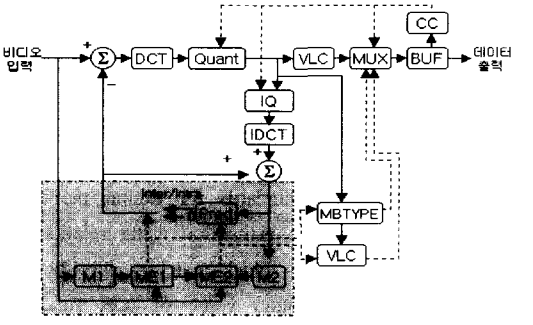


그림 1. H.263 엔코더 블록도

2.2 Nearest Neighbors 알고리즘

저 비트율 코딩과 실시간 화상 통신을 목적으로 움직임 예측에 필요한 계산량을 줄일 수 있도록 고안된 고속 탐색법 중에 H.263에서 권고한 탐색법이 Nearest Neighbors 탐색법이다. 그림 2에서 처럼 기준점에서 마름모 형태의 상하좌우 네 점(탐색 후보 화소)에 대해서 각각 16×16의 크기를 갖는 MB (macro block)내의 화소들의 차의 절대값들의 합인 SAD(Sum of Absolute Difference) 값을 계산한다. 그 중에서 최소 SAD 값을 갖는 점을 찾아 이 점을 새로운 기준점으로 설정하여, 다시 그 점의 상하좌우 네 점에 대해서 최소 SAD 값을 찾는다. 이와

같은 과정은 이전 단계에서 구한 최소 SAD 값이 현재 단계에서 구한 최소 SAD 값보다 작을 때까지 반복된다. 그림 2의 우측은 2단계까지 탐색하는 예를 보여준다. SAD를 계산하는 식은 다음과 같다.[3][4]

$$SAD(i,j) = \sum_{m=1,n=1}^{16,16} |CurImage_{m,n} - RefImage_{m+i,n+j}|$$

$$|i| < m_2,\ |j| < n_1$$

CurImagem,n : 현재 프레임의 (m, n)의 화소값
RefImagem,n : 과거 프레임의 (m, n)의 화소값
i, j : 탐색 후보 화소의 위치
m2, n1 : 탐색창의 가로축과 세로축 크기

SAD식의 변수는 위와 같이 나타낼 수 있다. 기준이 되는(m,n)위치의 MB내의 화소와 후보화소 (i,j) 위치의 MB내의 화소간의 SAD 연산을 적용한다.

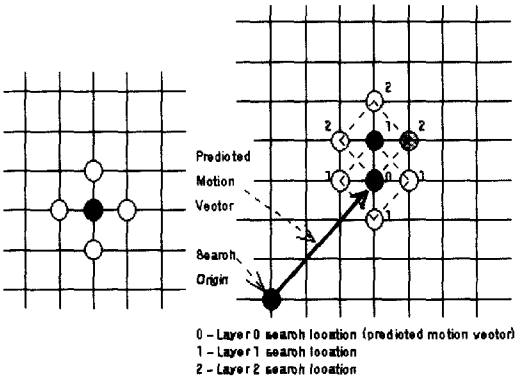


그림 2. Nearest Neighbors 탐색법

또한 Nearest Neighbors 탐색법은 시작점으로 영벡터(zero vector) 대신에 먼저 구한 인접 매크로블록들의 동작 벡터들의 중간 값(median value)을 예측 동작 벡터(predicted motion vector)로 사용함으로서 그 정확도를 더욱 높일 수 있다. 본 논문에서는 예측 동작 벡터를 사용하지 않고 시작점을 영벡터로 구현하였다.

Ⅲ. 하드웨어 설계

이 장에서는 2장에서 간단히 설명한 H.263 앤코더에 사용되는 Nearest Neighbors 탐색을 이용한 움직임 탐색 예측기의 구현에 대하여 설명한다.

현재 화상회의를 구현한 소프트웨어가 많이 나와 있으나 소프트웨어를 구동하는 시스템의 부하가 하드웨어로 구현한 시스템에 비하여 많이 걸린다. 실제로 소프트웨어로 실시간 화상회의를 하는 데에 있어서 화면이 끊기는 현상을 볼 수가 있다. 본 논문에서는 움직임 예측을 구현하고자 한다. 특히 움직임 예측에 전역 탐색 알고리듬 대신 Nearest Neighbors 탐색법을 이용한 고속 탐색 알고리듬을 사용하여 H.263코덱에 적용하고자 한다.

본 논문에서 구현한 고속 움직임 예측기의 시스템 구성은 그림 3과 같이 현재 프레임과 과거 프레임을 저장하고 있는 메모리를 인터페이스 할 수 있는 메모리 인터페이스 블록, SAD 연산을 하는 SAD 블록, 움직임 벡터를 생성하는 움직임 벡터 제너레이터 블록, 이것들을 제어하는 제어 블록으로 구성되어 있다.

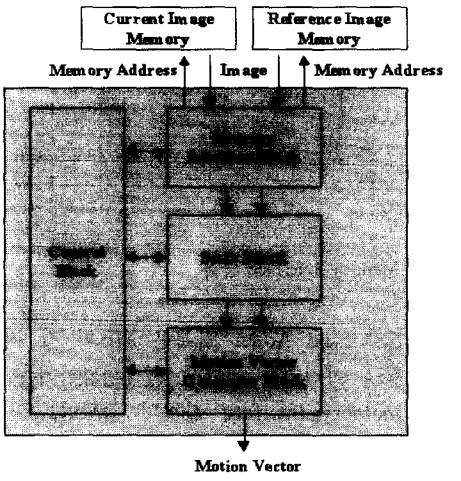


그림 3. 고속 움직임 예측기 구조

3.1 메모리 인터페이스 블록 설계

그림 4와 같이 Memory Interface 블록은 현재 영상이 저장된 RAM의 어드레스와 이 어드레스로부터 상하좌우 네 개의 후보화소를 가리키는 참조 영상이 저장된 RAM의 어드레스를 결정한다.

메모리에 영상이 저장되는 순서는 첫 번째 MB의 첫째 라인부터 열 한 번째 MB의 첫째 라인까지 순서대로 저장되어 진다. 그러나 최근에는 모든 영상을 블록 단위(16×16)로 처리를 한다. MB내에서의 다음 라인 화소값에 대한 어드레스를 액세스하기 위해선 현재 라인의 어드레스에, QCIF 파일의 넓이 화소인 176 만큼의 어드레스를 증가시켜야 한다. 이 과정을 16번째 라인까지 하게 되면 하나의 MB에

대한 메모리를 인터페이스하게 된다.

제어 블록으로부터 네 개의 후보 화소 중 최소 SAD를 갖는 화소가 결정되면 다시 이 어드레스를 갱신하여 기준 화소를 정하고 주위의 네 개의 후보 화소의 어드레스를 RAM으로 넘겨준다. 그리고 어드레스가 가리키는 RAM의 화소 데이터를 받아서 SAD 블록으로 넘겨 주게 된다. 이 과정을 현재 MB에 대한 움직임 벡터를 결정할 때까지 계속하게 된다.

Memory Interface의 가산기와 감산기에서는 기준 화소 주위의 4개의 후보 화소만을 가리키도록 1과 176만을 가·감산하도록 하여 불필요한 로직을 줄이고자 하였다.

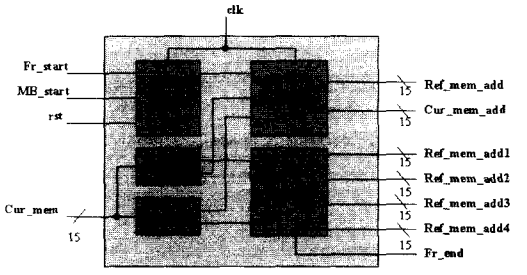


그림 4. Memory Interface 블록

그림 5의 Line Adder 블록은 MB의 각 라인의 첫 화소들을 가리키는 어드레스를 생성시키는 블록으로서 Memory Interface 블록의 컴포넌트로 사용되었다.

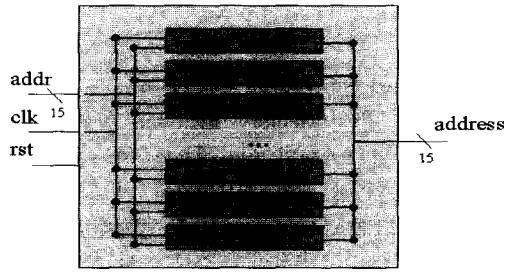


그림 5. Line Adder 블록

3.2 SAD 블록 설계

SAD 블록은 그림 6과 같이 구현하였다. Memory Interface 블록으로부터 화소값을 받은 후 현재 프레임의 화소값과 참조 프레임의 후보 영역의 화소값의 감산을 실행하는 감산기 블록과 감산기 블록에서 출력된 값을 절대값으로 바꾸는 Absolute 블록과 이 절대값들을 전부 합하는 가산기 블록과 기준 화소 주위의 4개의 후보화소들의 SAD 값만을 가지고서 비교하여 최소값을 넘기는 Comparison 블록을 가지고 SAD 블록의 하부 구조를 설계하였다.

이 블록은 하나의 기준화소에 대한 네 개의 후보 화소에 대해 SAD를 실행한다. 단 1번째, 11번째, 89번째, 99번째 MB는 처음 후보 화소를 두 개로 제한하고 SAD를 시작하고, 2~10번째, 90~98번째 MB는 처음 후보 화소를 세 개로 제한하도록 설계하였다. 그리고 SAD 연산 중에 프레임의 경계에서 넘지 않도록 경계에 도달하면 그 때의 최소 SAD값을 가지는 화소를 움직임 벡터로 출력하도록 하였다. Comparison 컴포넌트에서는 네 개의 후보 화소에 대한 SAD값들을 레지스터에 저장하였다가, 비교하여 그 중 최소 SAD 값을 결정하여 최소 SAD를 가진 후보화소의 위치와 최소 SAD 값을 Control 블록으로 출력하도록 구현하였다.

Control 블록에서는 전에 저장되어 있던 최소 SAD와 방금 SAD 블록에서 출력한 최소 SAD를 비교한 후, 새로운 기준점을 정할 것인지 SAD 연산을 멈출 것인지를 Control 블록에서 결정한다. 최소 SAD를 가지는 화소를 가리키는 어드레스는 Motion Vector Generator 블록으로 보내어 진다.

SAD 블록에서 컴포넌트로 사용된 감산기는 감수의 2의 보수화를 한 후, 부호 비트가 확장된 병렬 가산기로 구현하였다. 또한 산술 오버플로가 발생하면 양의 부호일 경우 최대값, 음의 부호일 경우 최소값을 출력하도록 설계하였다.

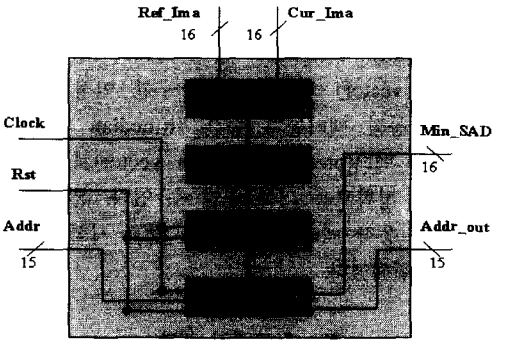


그림 6. SAD 블록

3.3 움직임 벡터 제너레이터 블록 설계

움직임 벡터 제너레이터 블록은 조건문, 카운터, 가산기와 감산기만으로 설계를 하였다. SAD 블록

에서 현재 MB에서의 최소 SAD 값을 가지는 화소를 결정하게 되면 해당 화소 위치를 가리키는 어드레스를 움직임 벡터 제너레이터 블록으로 넘겨주게 된다. 기준 화소에서부터의 어드레스와 최소 SAD 값을 가지는 화소의 어드레스의 차를 가지고 MB의 크기와 비교를 해서 움직임 벡터의 크기와 방향을 X좌표와 Y좌표의 벡터로 출력하는 블록이다. 동작 순서는 그림 7의 순서도에 나타내었다. 기준 화소보다 작은 경우는 가산으로, 큰 경우는 감산으로 연산을 한다. 이 때 마지막으로 더한 값에서 H.263 규격과 사용 가능하도록 ±15 화소 크기의 탐색영역 내에서 움직임 벡터를 탐색한다.

그림 8에는 움직임 벡터 제너레이터 블록을 나타내었다. 이 블록에서도 메모리 인터페이스 블록에서 사용된 가산기와 감산기 컴포넌트처럼 176만을 가·감산을 하도록 구현하였다. 카운터는 탐색창에서 어느 한 방향으로 최대 크기인 15번의 가·감산을 검사하도록 16진 카운터로 설계하였다.

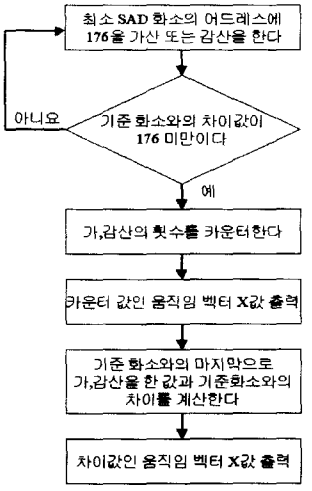


그림 7. 움직임 벡터 제너레이터 블록 동작 순서

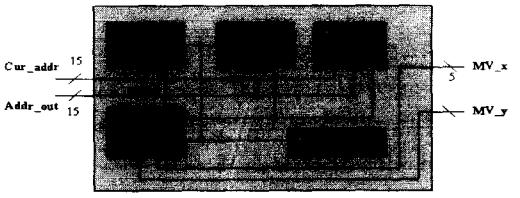


그림 8. 움직임 벡터 제너레이터 블록

3.4 제어 블록

그림 9의 제어 블록은 최소 SAD 값과 그 값의 어드레스, 몇 번째 MB에 대한 SAD 연산을 실행중인지를 알 수 있게 하는 카운터와 각 블록에 대한 클럭을 제공한다. Frame_Start 신호는 H.263코덱으로 확장 가능하도록 외부에서 입력을 받는 형태로 설계하였다. MB의 시작과 끝을 제어함으로써 MB단위로 처리하도록 하였다. 그리고 각 블록 간의 인터페이스를 제공한다.

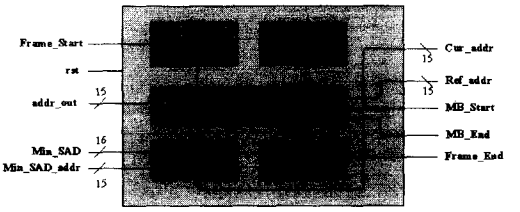


그림 9. 제어 블록

3.5 전체 시스템 구현

3.4절에서 설계한 각각의 블록들을 하위 구조로 포함시켜 컴포넌트화하여 톱 레벨에서는 각 블록들을 연결하는 방식으로 구조화하였다. 하드웨어는 VHDL을 이용하여 설계하고, Xilinx Foundation의 FPGA Express를 이용하여 설계를 검증하였다.

설계한 시스템의 입력 영상 포맷은 각 화소 값들은 16-bit의 폭을 가지고, QCIF 포맷을 처리할 수 있도록 설계하였다. 그리고 출력은 움직임 벡터의 크기가 ±15로 제한되도록 부호 비트가 포함된 5-bit 값을 출력한다. 그림 10에서는 고속 움직임 탐색 예측기의 전체 시스템을 보여주고 있다.

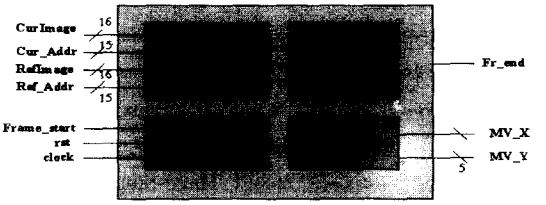


그림 10. 고속 움직임 예측기 블록도

그림 11에서는 본 시스템의 최상위 구조를 구현한 VHDL 코드를 보여준다. 각 블록은 컴포넌트로 선언하여 호출하는 Top-down 방식으로 설계하였다.

Ⅳ. 모의 실험 및 결과

4.1 모의 실험

그림 12의 영상은 본 논문에서 설계한 각 블록들과 전체 시스템의 시뮬레이션에 사용할 Foreman.

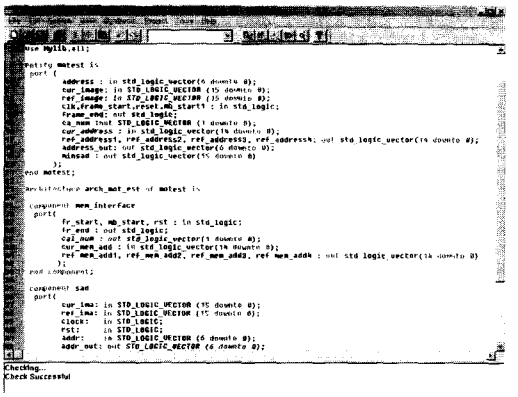


그림 11. 고속 움직임 예측기 VHDL 코드

qcif영상의 두 프레임 이며, 우측 그림의 중간의 사각형인 선택영역을 사용하였다. 특히 65번째 MB를 기준으로 시뮬레이션 하였다.

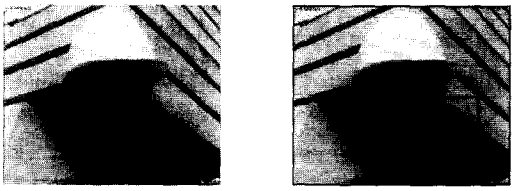


그림 12. Foreman 영상(좌: 참조 프레임 우: 현재 프레임)

표1은 현재프레임의 화소값들을, 표2는 참조 프레임의 화소값들을 나타내었다. 표에서 어두운 부분은 65번째 MB 영역을 나타낸다. 굵게 표시된 화소값은 65번째 MB에서 최초 기준점과 기준점에서의 후보 화소값을 나타낸다.

표 3에서는 기존의 Full-search 알고리즘 및 ThreeStep 알고리즘과 Nearest-Neighbors 알고리즘의 성능을 비교하기 위해 PSNR을 보인다. Nearest Neighbors알고리듬은 기존의 Three-Step 알고리즘보다 성능과 계산량 모두 우월하며 0.55dB 정도 더 나은 PSNR을 보인다.

표 1. Foreman 영상의 현재 프레임 화소값

70	71	69	69	59	54	49	49	45	49
61	57	**57**	57	62	64	57	50	49	51
77	110	127	125	110	88	65	54	49	49
92	112	152	174	171	155	116	73	52	49
107	141	172	190	196	182	149	110	68	50
118	157	190	199	194	180	152	111	73	52
111	151	165	158	170	176	157	117	74	51
81	118	171	186	178	162	142	110	71	48
67	91	109	147	162	144	94	65	46	36

표 2. Foreman 영상의 참조 프레임 화소값

68	61	**52**	48	47	47	44	47	48	47
54	**59**	**52**	**49**	50	48	50	47	50	47
95	78	**63**	**55**	51	47	49	48	49	49
169	156	**117**	**80**	54	50	47	47	44	47
185	181	**154**	**121**	**80**	53	45	48	44	46
204	193	**165**	**127**	**87**	**58**	44	46	47	48
170	183	**170**	**136**	**88**	**60**	45	46	46	48
178	163	**154**	**134**	**88**	**55**	45	41	44	48
172	174	**140**	**88**	53	45	39	40	46	44

표 3. 각 알고리즘의 성능 및 계산량 비교

Algorithm	PSNR(dB)	Matched MV	|·|, +/-
FS	29.22	100%	5.77E+09
TS	28.20	56.4%	2.44E+08
NN	28.75	72.1%	1.36E+08

4.2 메모리 인터페이스 블록 모의 실험

메모리 인터페이스 블록에서는 기준 어드레스와 mb_start신호를 입력으로 받았을 때, 참조 프레임이 저장된 메모리를 인터페이스하는 ref_add1, ref_add2, ref_add3, ref_add4에서 4개의 후보화소를 가리키는 어드레스가 출력되게 된다. 그림 13에는 현재의 기준 화소의 어드레스가 입력되면 주위의 4개의 후보 화소의 어드레스(ref_add1, ref_add2, ref_add3, ref_add4)가 출력되는 것을 볼 수 있다.

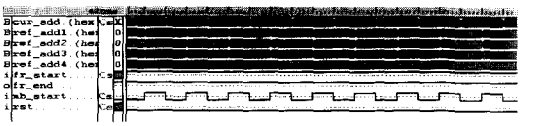


그림 13. Memory Interface 블록 시뮬레이션

4.3 SAD 블록 모의 실험

SAD 블록에서는 두 개의 화소값과 기준 화소를 둘러싼 후보 화소의 위치를 입력받는다. 네 개의 화소 중에서 최소 SAD 값과 네 개의 화소 중에서 어느 화소가 최소 SAD를 가지는지 그 위치를 출력하게 된다. 그림 14를 보면 현재 프레임의 기준 화소(CurImage)와 참조 프레임의 후보 화소(RefImage)가 순서대로 입력되며, 그 중에 최소 SAD값(sad)과 그 값을 가지는 화소의 어드레스(po_out)를 출력하는 것을 볼 수가 있다.

그림 14. SAD 블록 시뮬레이션

4.4 움직임 벡터 제너레이터 블록 모의 실험

Motion Vector Generator 블록은 최소 SAD 값을 가지는 화소의 어드레스와 현재 MB의 기준 화소의 어드레스를 입력으로 받아서 움직임 벡터를 X축, Y축을 기준으로 방향과 거리를 가리키는 값을 출력한다. 그림 15에서 보면 기준 화소의 어드레스(cur_addr)와 최소 SAD 값을 가진 화소의 어드레스(ref_addr)를 입력을 받은 후에 거리와 방향을 각각 mv_X(수평)와 mv_Y(수직)로 출력하는 것을 볼 수 있다.

그림 15. 움직임 벡터 제너레이터 블록 시뮬레이션

4.5 제어 블록과 전체 시스템 모의 실험

제어 블록은 블록 자체 시뮬레이션을 하지 않고, 전체 시스템과 같이 실험을 하였다. 그림 16에서는 고속 움직임 탐색 전체 블록에 대한 시뮬레이션이다. MB에서 처음 시작하게 되면 현재 영상의 화소값과 참조 영상의 화소값을 가지고서 최소 SAD를 찾을 때까지 수행 후에 벡터값(mv_x, mv_y)을 출력하는 것을 볼 수가 있다.

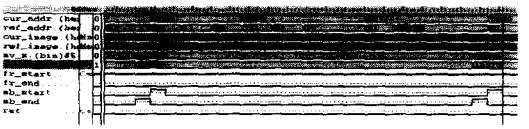


그림 16. 전체 시스템 시뮬레이션

4.6 결과

시스템을 구현한 타겟 디바이스에 대한 정보는 다음과 같다.

- Vendor : Xilinx · Family: VIRTEX
- Device : V300PQ240 · Speed: -6
- Optimize for : Speed
- Optimization effort: Low
- Frequency : 50 MHz

본 논문에서 구현한 고속 움직임 탐색 예측기를 Xilinx사의 Virtex 칩에서의 Floor Planning을 그림 17에서 보여준다.

그림 17에서는 구현한 시스템을 FPGA 칩에서 Xilinx의 Timing Analyzer를 가지고 Pin-to-Pin 딜레이를 보여주고 있다.

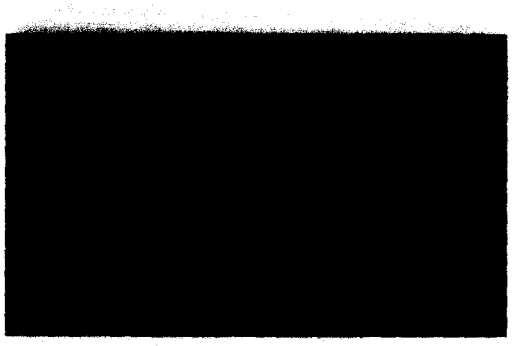


그림 17. 시스템 Floor Planning

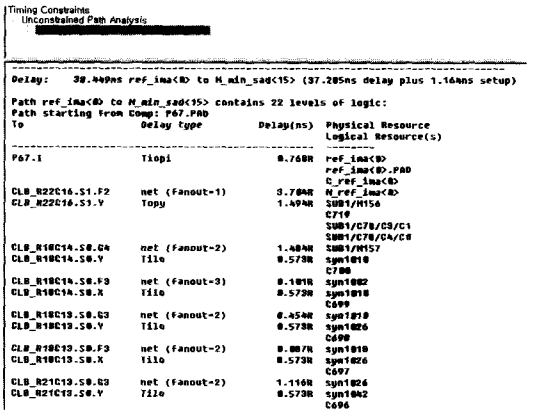


그림 18. 시스템 타이밍 분석

표 4에는 고속 움직임 예측기를 Xilinx FPGA 타겟 디바이스에서의 MB와 프레임에 대한 타이밍 분석과 자원 이용률의 결과를 보여주고 있다.

표 4. 고속 움직임 예측기의 FPGA Design 요약

Minimum Period (MB)	52.906 ns
Minimum Period (Frame)	15.871 ms
Total equivalent gate count	6912 개
Slice Flip Flop	194 개
4 Input LUTs	474 개

그림 19는 설계한 시스템을 구현한 FPGA 타겟 보드를 보여주고 있다.

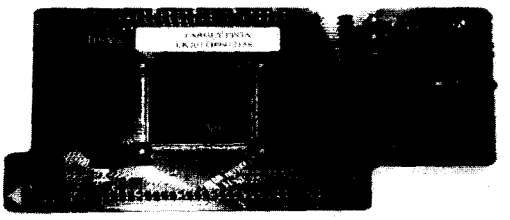


그림 19. 타겟 FPGA 보드

V. 결 론

본 논문에서는 화상회의, 화상통신에서 실시간으로 구현하는데 어려움이 많은 움직임 예측을 H.263을 모델로 삼아 VHDL을 이용하여 하드웨어로 구현하였다. 블록 정합 알고리듬을 이용하는 움직임 예측에 필요한 움직임 벡터를 고속으로 찾는 알고리듬 중에서, ITU-T에서 H.263 규격에 사용할 수 있는 고속 탐색 알고리듬으로서 Nearest Neighbors 탐색 알고리듬을 권고하였다. 이 알고리듬을 이용하여, 고속 움직임 탐색 예측기를 구현하였다. 전체적인 구성 방식은 Top-down 방식으로 설계 및 시뮬레이션 하였고, 설계한 본 시스템의 동작이 Xilinx FPGA(XCV300PQ240)에서 이상없이 동작함을 모의 실험과 Timing Analyzer를 통하여 확인할 수 있었다. 본 논문에서 구현된 블록의 성능은 표 4에 보였듯이, 전체 6912개의 게이트를 사용하였다. Nearest Neighbors 탐색법의 특징인 고속탐색으로 인해 하나의 MB를 처리하는데 52.9ns의 시간만이 소요되었다. 이는 화상통신을 하는 데에 전혀 장해를 받지 않는 초당 30프레임 이상의 성능을 보여줄 수 있다. 또 본 논문에서 구현된 블록은 H.263 엔코더 권고안에 맞추어 설계를 하여 추후 H.263 코덱에 적용할 수 있으리라 사료된다.

향후 과제로는, ITU에서 전화망을 이용한 화상단말기의 표준안으로 H.324를 확정 발표하였는데, 이에 대한 구성은 H.263(비디오), G.723(오디오), H.223(멀티프렉서), H.245(시스템 제어) 등으로 구성되어 있다. H.324 시스템의 비디오 Codec에 해당하는 H.263을 설계하기 위해, 화면내 부호화(Intra Frame Coding) 모듈과 본 논문에서 설계한 고속 움직임 탐색기와 같이 설계를 한다면 구현이 가능할 것으로 사료된다.

참 고 문 헌

[1] Borko Furht, Joshua Greenberg, and Raymond Westwater, "MotionEstimation Algorithms for Video Compression", Kluwer Academic Publishers, 1997

[2] K. R. Rao and J. J. Hwang, "Techniques & Standards for Image · Video & Audio Coding", Prentice Hall, 1996

[3] Guy Cote, Michael Gallant and Faouzi Kossentini, "Efficient Motion Vector Estimation and Coding for H.263-Based Very Low Bit Rate Video Compression", IEEE Transaction on Circuits and System for video technology vol.8 . no7. nov. 1998

[4] Guy Cote, Michael Gallant and Faouzi Kossentini, "An Efficient Computation-Constrained Block-Based Motion Estimation Algorithm for Low Bit Rate Video Coding", IEEE Transactions on Image Processing , V.8 N.12 , 1816-1823 , 19991201

[5] David Van Den Bout, "The Practical XILINX Designer Lab Book", Prentice-Hall

[6] Charles H. Roth, Jr. "Digital System Design Using VHDL", PWS Publishing Company

[7] S. K. Azim, "A low cost application specific video codec for consumer videophone," IEEE 1994 Custom Integrated Circuits Conference, San Diego, CA, pp.115-118, May 1994

[8] Y. Ninomiya and Y. Ohtsuka, "A motion compensated interframe coding scheme for television signals", IEEE Trans. commun., vol. COM-32, pp.328-334, Mar.1984

[9] R. W. Young and N. G. Kingsbury, "Video compression using lapped transforms for motion estimation/compensation and coding," Optical Engineering, vol. 7, pp.1451-1463, July 1993

[10] ITU-T Telecom. Standardization Sector of ITU, "Video Coding for Low Bit Rate Communication", Draft ITU-T Recommendation H.263, May 1996

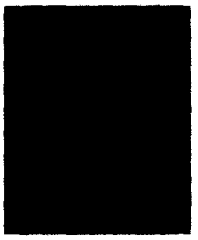
[11] ITU Telecom. Standarization Sector of ITU, "Video Codec Test Model Near-Term, Version5 (TMN5)," H.263 Ad Hoc Group, Jan. 1996

[12] Michel Bret, "Image Synthesis", Kluwer

Academic Publishers, 1992
[13] Majid Rabbani, Paul W. Jones, "Digital Image Compression Techniques", SPIE, 1991

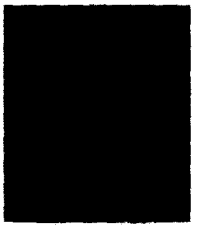
김 진 연(Kim Jin Yean)
1999년 2월 : 호서대학교 정보통신공학과 졸업
2001년 2월 : 호서대학교 정보통신공학과 석사
2001년~현재 : 버사칩스 근무
<주관심 분야> 영상 관련 칩 설계

박 상 봉(Park Sang Bong)



1985년 2월 : 광운대학교 전자재료공학과 졸업
1987년 2월 : 고려대학교 전자공학과 석사
1992년 2월 : 고려대학교 전자공학과 박사
1994년 3월~1999년 2월 : 삼성반도체 ASIC 설계 팀
2000년 7월~현재 : @Lab Digital Design Team
2000년 3월~현재 : (주)옴미디어 ASIC Team
1999년 3월~현재 : 세명대학교 정보통신학과 조교수.
<주관심 분야> Digital TV, Embedded Memory Test, Serial ATA.

진 현 준(Jin Hyun Jun)



1984년 2월 : 고려대학교 전자공학과 졸업
1986년 2월 : 고려대학교 전자공학과 석사
1986년 3월~1991년 6월 : 삼성전자 시스템개발실
1992년 9월~1998년 1월 : 미국 리하이대학교 전산학 박사
1998년 3월~현재 : 호서대학교 전기정보통신공학부 조교수
<주관심 분야> 영상처리, 시스템소프트웨어 등.

박 노 경(Park Nho Kyung)

1984년 2월 : 고려대학교 전자공학과 졸업
1986년 2월 : 고려대학교 전자공학과 석사
1990년 2월 : 고려대학교 전자공학과 공학박사
1996년 3월~현재 : (주)옴니미디어 기술고문
1999년 3월~2000년 2월 : OSU ECE 연구교수
2000년 12월~현재 : IDEC 호서대학교 책임교수
1989년 4월~현재 : 호서대학교 전기정보통신공학부 교수
<주관심 분야> HDTV, Image Processing Algorithm 및 ASIC Design