실시간 윈도우 기반 영상 처리를 위한 병렬 하드웨어 구조의 FPGA 구현

진 승 훈[†] · 조 정 욱[††] · 권 기 호[†††] · 전 재 욱[††††]

요 약

윈도우 기반의 영상처리는 전체 영상처리 분야에 있어서 기본이 되는 분야이다. 이러한 윈도우 기반의 영상처리는 처리해야 할 데이터와 연산이 매우 많은 편이기 때문에 범용 컴퓨터 구조에서 소프트웨어 프로그램을 사용하여 윈도우 기반 영상처리에서 필요로 하는 모든 연산을 실시간으로 수행하기 힘들다.

본 논문에서는 FPGA(Field Programmable Gate Array)를 사용하여 윈도우 기반 영상처리를 실시간으로 수행할 수 있는 병렬 하드웨어 구조를 제안하고자 한다. 또한 제안한 구조를 통해 VHDL(VHSIC Hardware Description Language)을 이용하여 윈도우 기반의 영상처리 중 하나인 동적 문턱치화(dynamic thresholding) 회로와 국부 히스토그램 평활화(local histogram equalization) 회로를 설계하고 FPGA로 해당 회로를 구현할 것이다. 구현된 회로의 성능 측정도 다루어 진다.

키워드 : 윈도우 기반 영상처리, 동적 문턱치화, 국부 히스토그램 평활화, FPGA, VHDL

An FPGA Implementation of Parallel Hardware Architecture for the Real-time Window-based Image Processing

S. H. Jin[†] · J. U. Cho[††] · K. H. Kwon[†††] · J. W. Jeon[††††]

ABSTRACT

A window-based image processing is an elementary part of image processing area. Because window-based image processing is computationally intensive and data intensive, it is hard to perform all of the operations of a window-based image processing in real-time by using a software program on general-purpose computers.

This paper proposes a parallel hardware architecture that can perform a window-based image processing in real-time using FPGA (Field Programmable Gate Array). A dynamic threshold circuit and a local histogram equalization circuit of the proposed architecture are designed using VHDL(VHSIC Hardware Description Language) and implemented with an FPGA. The performances of both implementations are measured.

Key Words : Window-Based Image Processing, Dynamic Threshold, Local Histogram Equalization, VHDL, FPGA

1. 서 론

영상처리는 여러 분야에서 응용되고 있으며 일반적으로 많은 양의 데이터와 연산을 필요로 한다. 가장 기초적인 영상처리 방법으로 픽셀 기반 처리(Pixel Point Processing)와 픽셀 그룹 처리(Pixel Group Processing)를 들 수 있다. 한 픽셀의 값을 변환할 때 다른 픽셀에 관계없이 현재 픽셀의 값만을 사용하여 변환하는 픽셀 기반 처리와 달리 픽셀 그룹 처리는 출력 영상의 새로운 픽셀 값을 결정하기 위해 해당 픽셀과 그 주변 픽셀을 함께 고려한다. 한 픽셀의 주변 픽셀들은 처리 영역 내의 국부적인 밝기 성향에 대한 정보를 제공해 줄 수 있기 때문에 픽셀 기반 처리에서 얻을 수 없는 정보를 획득하는 것이 가능하다.

픽셀 그룹 처리는 크게 선형 공간 필터(Linear Spatial Filter)와 비선형 공간 필터(Non-linear Spatial Filter)로 나눌 수 있다. 선형 공간 필터는 현재 픽셀 값을 이미 정해진 가중치 값을 가지고 있는 2차원 배열을 현재 픽셀과 이웃하는 픽셀에 곱하여 더한 값으로 변경한다. 이때 사용되는 2

※ 이 연구는 산업자원부 지원으로 수행하는 21세기 프론티어 연구개발사업 (인간기능생활지원지능로봇 기술개발사업)의 일환으로 수행되었습니다.
† 준 회 원 : 성균관대학교 전기전자공학과 석사과정
†† 준 회 원 : 성균관대학교 정보통신공학부 연구교원
††† 정 회 원 : 성균관대학교 정보통신공학부 교수
†††† 종신회원 : 성균관대학교 정보통신공학부 교수
논문접수 : 2005년 10월 7일, 심사완료 : 2006년 4월 3일

차원 배열을 윈도우(Window) 또는 커널(Kernel)이라 부른다. 비선형 공간 필터는 윈도우와의 선형적인 합으로 나타내는 것이 불가능한 이웃 픽셀들과의 비선형적인 연산에 기반을 둔다. 선형 공간 필터와 비선형 공간 필터 모두 여러 가지 모양의 윈도우를 한 픽셀과 그 주변 픽셀에 적용한 결과를 이용한다. 이처럼 윈도우를 사용한 픽셀 그룹 처리를 윈도우 기반의 영상 처리(Window-based Image Processing)라고 부른다[1].

이러한 윈도우 기반의 영상 처리는 다른 영상처리 방법의 전처리 과정으로 많이 사용되며 그 특성상 한 픽셀과 그 주변 픽셀을 동시에 접근 가능한 방법을 제공하기 때문에 다른 여러 영상처리의 일부분으로도 자주 사용된다. 전처리 과정으로 사용되는 대표적인 예로서는 LOG(Log Of Gaussian) 필터와 저주파/고주파 통과 필터 등이 있으며, 다른 영상처리의 일부분으로 사용되는 경우는 국부 히스토그램 평활화와 스테레오 비전 등을 들 수 있다.

윈도우 기반 영상처리는 픽셀 기반 처리와 달리 한 픽셀과 그 주변의 픽셀 모두를 사용하여 연산을 하게 되며 이와 같은 연산이 전체 영상을 이루는 각 픽셀에 대해 반복적으로 수행하게 된다[2]. 저장 프로시저 방식을 따르는 범용 컴퓨터의 경우 이러한 윈도우 기반의 영상 처리를 수행하기 위해서는 메모리에 저장된 각 픽셀 값을 반복해서 CPU로 읽어와야 한다. 예를 들어 $M \times N$ 해상도를 가지는 2D 영상에서 $m \times n$ 크기의 윈도우를 사용하여 윈도우 기반의 영상처리를 적용한다고 가정하면 이때의 시간복잡도(Time Complexity)는 최소한 $O(m \times n \times M \times N)$ 보다 크게 된다[3]. 이는 하나의 2D 영상에 대해서 윈도우 기반의 영상 처리를 위해 필요한 메모리 접근의 수가 캐쉬 최적화 등을 고려하지 않았을 때 최소 $m \times n \times M \times N$번 이상이기 때문이다.

또한 정지영상이 아닌 동영상의 경우에는 처리해야 할 데이터량과 연산량이 더욱 크게 늘어나게 된다. 예를 들어 프레임 전송률(frame rate)이 30 fps(frames per second)이고 해상도가 640×480인 RS-170 표준 영상을 입력으로 받고 이에 5×5크기 윈도우를 사용하여 윈도우 기반의 영상처리를 적용하는 경우에는 1초 동안 2억3천만번 이상의 메모리 참조가 요구된다. 일반적으로 범용 마이크로 프로세서 에서는 명령 수행에 걸리는 시간보다 메모리를 참조 하는데 걸리는 시간이 더 길기 때문에 이처럼 과다한 메모리 참조는 전체 영상처리에 있어 병목현상을 만들고, 실시간 처리를 방해하는 주요 원인이 된다[4].

본 논문에서는 이와 같은 메모리 참조에 의한 병목현상을 해결하여 윈도우 기반 영상처리를 실시간으로 수행하기 위한 병렬 하드웨어 구조를 제안한다. 제안한 구조에서 영상을 이루는 각 픽셀에 대해 해당 픽셀과 주위 픽셀들은 특정 레지스터에 저장되며, 이 레지스터에 저장된 픽셀들은 모두 동시 참조가 가능하다. 그리고 이렇게 동시 접근 가능한 픽셀들을 파이프라인 기반으로 각 응용에 따른 영상 처리를 적용하여 윈도우 기반 영상처리를 실시간으로 수행할 수 있게 된다.

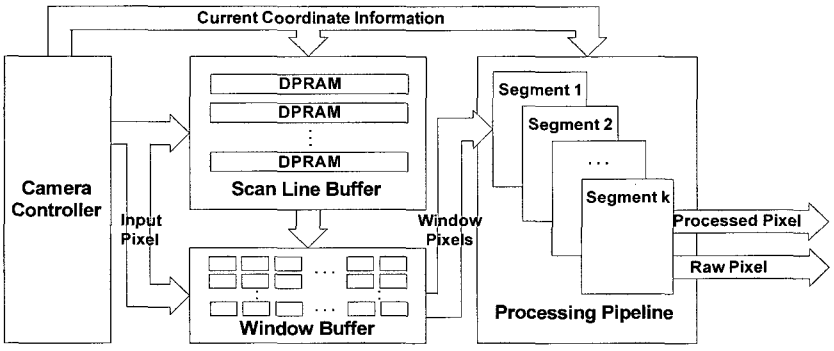
본 논문은 다음과 같이 구성된다. 2장에서는 윈도우 기반 영상처리를 실시간으로 수행할 수 있는 병렬 하드웨어 구조를 제안한다. 3장에서는 제안한 구조를 사용하여 동적 문턱치화 회로와 국부 히스토그램 평활화 회로를 설계/구현하고 4장에서 실험 결과에 대해 살펴본다. 마지막으로 5장에서는 결론을 기술한다.

2. 실시간 윈도우 기반 영상처리 구조

위에서 언급한 바와 같이 윈도우 기반 영상처리의 특징으로 많은 양의 데이터 접근이 필요하다는 점과 비교적 간단한 연산이 영상을 이루는 각 픽셀에 대해 반복적으로 수행된다는 점을 들 수 있다[5]. 본 논문 에서는 이러한 특징을 만족시키면서 실시간으로 윈 도우 기반의 영상처리를 수행하기 위해서 (그림 1)과 같은 병렬 하드웨어 기반의 영상처리 구조를 제안한다.

제안한 실시간 윈도우 기반 영상처리 구조는 크게 스캔라인 버퍼(scan-line buffer), 윈도우 버퍼(window buffer), 연산 파이프라인(processing pipeline)의 세 부분으로 구성된다. 제안한 구조의 동작을 설명하기 위해 카메라에서 출력되는 2D영상의 해상도가 640×480이고, 윈도우의 크기는 5×5라고 가정한다.

2.1 픽셀 이동



(그림 1) 제안한 윈도우 기반 영상처리 구조
(Fig. 1) Proposed window-based image processing architecture

실시간으로 윈도우 기반 영상처리를 수행하려면 윈도우에 해당하는 각 픽셀 값을 얻을 때 메모리 접근에 따른 시간 지연을 최소화 할 필요가 있다. 제안한 구조에서는 스캔라인 버퍼와 윈도우 버퍼를 사용하여 윈도우에 해당하는 픽셀을 항상 레지스터에 유지 함으로써 픽셀 접근에 따른 시간 지연을 최소화 시킨다.

스캔라인 버퍼는 *4*개의 DPRAM(Dual Port Random Access Memory)으로 구성되며 각 DPRAM은 각 원소가 픽셀의 값이며 입력 영상의 현재 *x*좌표 i_c를 주소로 가지는 일차원 배열과 같다. 윈도우 버퍼는 *5×5*개의 레지스터로 구성되며 각 레지스터는 하나의 픽셀 값을 저장한다.

카메라에서 입력 받는 각 픽셀이 어떻게 스캔라인 버퍼와 윈도우 버퍼를 통해 주변 픽셀과 함께 레지스터에 저장되는지 보이기 위해 *640×4* 크기의 스캔라인 버퍼 *L*과 *5×5* 크기의 윈도우 버퍼 *W*를 예로 설명한다. (i_c,j_c) 번째 픽셀 $p(i_c,j_c)$을 카메라에서 입력 받을 때 윈도우 버퍼와 스캔라인 버퍼의 각 픽셀은 아래 식 (1)-(10)에 나타난 바와 같이 이동된다.

$$\begin{aligned} W(i\text{-}1,1) &\rightarrow W(i\text{-}1,1) \\ W(i\text{-}1,2) &\rightarrow W(i\text{-}1,2) \\ &\cdots \\ W(i\text{-}1,5) &\rightarrow W(i\text{-}1,5) \end{aligned} \qquad \text{식 (1)}$$

여기서 i는 $2 \le i \le 5$의 범위를 갖는다.

$$W(5,1) \rightarrow L(i_c,1) \qquad \text{식 (2)}$$

$$L(i_c,1) \rightarrow L(i_c,2) \qquad \text{식 (3)}$$

$$W(5,2) \rightarrow L(i_c,2) \qquad \text{식 (4)}$$

$$L(i_c,2) \rightarrow L(i_c,3) \qquad \text{식 (5)}$$

$$W(5,3) \rightarrow L(i_c,3) \qquad \text{식 (6)}$$

$$L(i_c,3) \rightarrow L(i_c,4) \qquad \text{식 (7)}$$

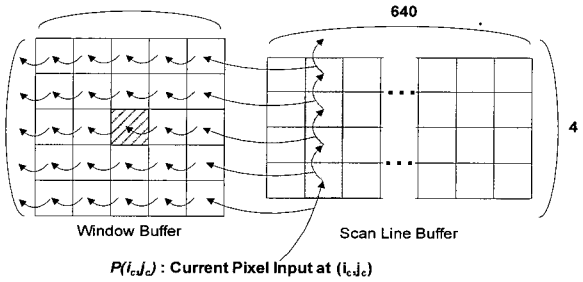
$$W(5,4) \rightarrow L(i_c,4) \qquad \text{식 (8)}$$

$$L(i_c,4) \rightarrow p(i_c,j_c) \qquad \text{식 (9)}$$

$$W(5,5) \rightarrow p(i_c,j_c) \qquad \text{식 (10)}$$

(그림 2)는 *640×4* 크기의 스캔라인 버퍼와 *5×5* 크기의 윈도우 버퍼에서 픽셀 값의 이동을 보여준다. (그림 3)과 (그림 4)는 각각 *640×4* 크기의 스캔라인 버퍼와 *5×5* 크기의 윈도우 버퍼 내 픽셀 값의 이동과 관련된 VHDL 구문을 보인다. (그림 3)의 process 구문은 식 (1)에서 *i=5*일 때의 동작을 수행하며 이와 같은 연산을 통해 매 동기 클럭마다 윈도우 버퍼의 각 픽셀 값을 현재 입력중인 픽셀을 기준으로 재구성하는 것과 같은 결과를 얻는다. 스캔라인 버퍼를 구성하는 각 DPRAM은 i_c를 주소로 사용하기 때문에 *x*좌표가 현재 입력중인 픽셀과 동일한 각 픽셀들을 윈도우 버퍼와 상위 DPRAM에 복사하여 입력중인 라인과 상위 *4*라인을 항상 유지하게 된다.

이러한 각 연산은 VHDL의 signal이 델타 지연(delta delay) 특성을 이용하여 병렬적으로 수행되며 윈도우 버퍼는 최초 픽셀이 입력된 후 *2*라인, *2*픽셀의 지연 후에 $p(i_c\text{-}2,\ j_c\text{-}2)$을 중심으로 항상 *5×5*개의 픽셀을 레지스터에 유지하게 된다.



(그림 2) 윈도우/스캔라인 버퍼에서 픽셀의 이동
(Fig. 2) Transfer of pixels on window and scan-line buffer

```
WINDOW4_HANDLER:
process(PIXEL_CLK)
begin
    if falling_edge(PIXEL_CLK) then
        Window(4,1) <= Window(5,1);
        Window(4,2) <= Window(5,2);
        Window(4,3) <= Window(5,3);
        Window(4,4) <= Window(5,4);
        Window(4,5) <= Window(5,5);
    end if;
end process;
```

(그림 3) 윈도우 버퍼 제어 process 구문
(Fig. 3) process statement for controlling the window buffer

```
LINE1_HANDLER:
process(PIXEL_CLK, LINE_SHIFT, Input_X)
begin
    if (LINE_SHIFT = '1') then
        Line1_AddrB <= conv_std_logic_vector(1, 10);
    elsif falling_edge(PIXEL_CLK) then
        Line1_AddrB <= conv_std_logic_vector(Input_X+1, 10);
    end if;
    Window(5,1) <= conv_integer(Line1_DoutB);
end process;

LINE2_HANDLER:
process(PIXEL_CLK, LINE_SHIFT, Input_X)
begin
    if (LINE_SHIFT = '1') then
        Line1_DinA <= (others => '0');
        Line1_AddrA <= (others => '0');
        Line2_AddrB <= conv_std_logic_vector(1, 10);
    elsif falling_edge(PIXEL_CLK) then
        Line1_DinA <= Line2_DoutB;
        Line1_AddrA <= conv_std_logic_vector(Input_X, 10);
        Line2_AddrB <= conv_std_logic_vector(Input_X+1, 10);
    end if;
    Window(5,2) <= conv_integer(Line2_DoutB);
end process;

LINE5_HANDLER:
process(PIXEL_CLK, LINE_SHIFT, Input_X)
begin
    if (LINE_SHIFT = '1') then
        Line4_DinA <= (others => '0');
        Line4_AddrA <= (others => '0');
    elsif falling_edge(PIXEL_CLK) then
        Line4_DinA <= CURRENT_PIXEL;
        Line4_AddrA <= conv_std_logic_vector(Input_X, 10);
    end if;
    Window(5,5) <= conv_integer(CURRENT_PIXEL);
end process;
```

(그림 4) 스캔라인 버퍼 제어 process 구문
(Fig. 4) process statements for controlling the scan-line buffer

2.2 파이프라인 처리

윈도우 기반의 영상처리를 실시간으로 수행하기 위해서 고려해야 할 다음 사항으로 실제 윈도우 연산부분을 들 수 있다. 윈도우 연산은 영상을 이루는 각 픽셀이 카메라에서 동일한 간격으로 생성되어 순차적으로 전달된다는 점과 비교적 단순한 연산이 각 픽셀에 대해 반복된다는 점에서 파이프라인 기법을 적용하는 것이 적절하다. (그림 1)에서 k개의 세그먼트로 구성된 연산 파이프라인을 보인다.

윈도우 기반 영상처리를 파이프라인 기법으로 처리하려면 수행해야 할 영상처리 동작을 복잡도가 유사한 세그먼트로 분리하여야 한다. 세그먼트의 개수는 수행하고자 하는 윈도우 기반 영상처리 복잡도와 파이프라인 클럭의 주기를 고려하여 결정할 수 있다. 윈도우 버퍼를 이루는 각 레지스터의 값은 각 픽셀이 입력될 때 마다 변경되기 때문에 가장 큰 세그먼트의 수행시간을 픽셀 입력 주기보다 짧도록 해야 한다. 따라서 영상처리의 복잡도가 높으면 세그먼트의 수를 늘려야 하고 파이프라인 클럭의 주기가 짧으면 세그먼트의 수를 줄일 수 있다.

제안한 윈도우 기반 영상처리 구조는 입력과 출력 모두를 카메라에서 입력 받는 픽셀과 동기화 하기 때문에 특정 응용을 위해 직렬 또는 병렬로 연결하여 사용하는 것이 가능하다. 이때 전체 회로의 지연 시간은 직렬로 연결한 경우에는 각 모듈의 지연시간을 합한 것과 같고 병렬로 연결한 경우에는 가장 긴 지연시간을 가지는 모듈과 같게 된다.

3. 영상 처리 회로 구현

3.1 동적 문턱치화

전역(global) 문턱치화는 전체 영상에 단일 문턱 값을 적용하여 배경(background)과 전경(foreground) 정보를 구분하는 간단한 방법이다. 이 방법은 영상을 이루는 전체 픽셀의 그레이 값과 단일 문턱 값을 비교하기 때문에 적절한 문턱 값을 선택하는 것이 중요하다. 그런데 영상의 부분적인 영역에 걸쳐서 밝기 변화가 있을 경우에는 배경 값이 균일해지지 않기 때문에 영상의 각 픽셀을 배경과 전경으로 적절하게 분류하기 힘들다.

이러한 문제를 해결하기 위해서 각 픽셀 별로 적절한 문턱 값을 적용하는 방법이 필요하다[7]. 각 픽셀 별로 문턱 값을 계산하는 방법에는 Chow 방법과 국부 문턱치화 방법의 두 가지가 있다. 두 방법 모두 영상을 조명이 일정한 하위 영역으로 세분하고 각 영역 별로 적절한 문턱 값을 찾아내는 방법을 사용한다[8].

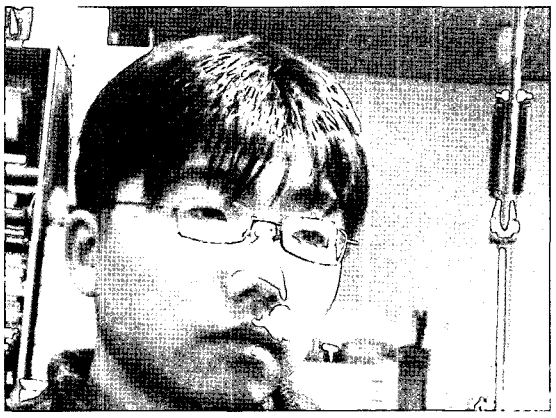
Chow 방법은 영상을 일정한 크기의 작은 영역으로 나눈 후 각 영역별로 히스토그램을 계산하여 이를 통해 해당 영역의 문턱 값을 결정한다. 이 방법은 나누어지는 영역의 크기가 최종 결과 영상에 큰 영향을 주며 영역을 작게 나눌수록 그만큼 히스토그램 계산 회수의 증가에 따른 부담이 커지게 되어 실시간 처리가 어려워 진다.

국부 문턱치화 방법은 각 픽셀에 대해 주위 픽셀을 고려하여 해당 픽셀에 적용될 문턱 값을 결정하는 방법이다. 대표적으로 한 픽셀에 대해 주변 픽셀의 평균값(average)이나 중앙값(median)과 대소를 비교하여 주변 픽셀과 얼마나 유사한지 판단, 해당 픽셀의 전경/배경 여부를 구분하는 방법을 예로 들 수 있다. 국부 문턱치화 방법에서도 각 픽셀 별로 비교의 대상이 되는 주위 픽셀의 범위를 선택하는 것이 결과 영상에 큰 영향을 준다. 너무 많은 주변 픽셀을 포함시키면 전역 문턱치화와 같이 국부적인 밝기의 변화가 결과 영상에 영향을 미칠 수 있으며 반대로 주변 픽셀을 너무 적게 포함시키면 영상에 포함된 잡음 때문에 잘못 계산된 문턱 값을 적용하게 된다.

3.2 동적 문턱치화 구현

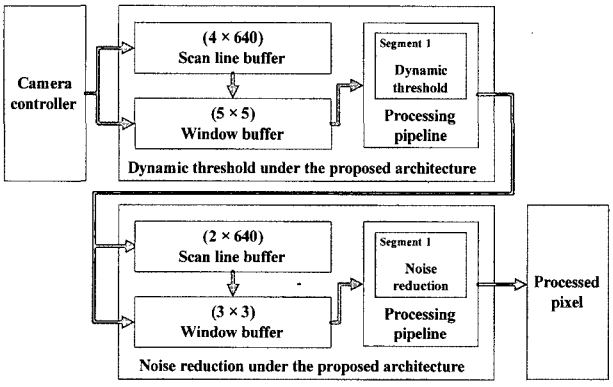
국부 문턱치화 방법은 한 픽셀과 그 주변 픽셀을 이용하여 각 픽셀에 적용될 문턱 값을 계산하는 방법이기 때문에, 본 논문에서 제안한 윈도우 기반의 영상처리 구조를 적용하기 적합하다. 각 픽셀에 대한 문턱 값은 주변 5×5 윈도우에 속하는 픽셀들의 그레이 레벨 평균값으로 정하였고 윈도우 크기는 실험을 통해 결정하였다. 국부 문턱치화에서 문턱 값을 구하는 것은 한 픽셀 주기 내에 처리가 가능한 연산이므로 파이프라인에서 단일 세그먼트만 사용해도 실시간으로 처리하는데 지장이 없다. (그림 5)에 원본 실험 영상을, (그림 6)은 실험 영상에 동적 문턱치화를 적용한 결과를 나타낸다.

동적 문턱치화를 적용한 결과 영상은 잡음이 많이 나타나는데, 잡음 감소 필터를 적용하여 대부분의 잡음을 감소시킬 수 있다. 본 논문에서는 한 픽셀에 대해서 3×3 윈도우를 설정하고 이 윈도우 내에 검은색의 픽셀이 3개 이하이면 해당 픽셀을 잡음으로 인식하고 제거하는 잡음 감소 필터를 사용하였다. 이러한 잡음 제거 필터 역시 윈도우 연산을 기반으로 하기 때문에 본 논문에서 제안한 구조를 이용하여 구현이 가능하다. 또한 동적 문턱치화를 적용한 영상을 잡음 감소 필터의 입력으로 주어야 하기 때문에 (그림 7)에 나타난 바와 같이 동적 문턱치화 회로에 직렬로 연결하여 사용하였다. 원본 영상의 첫 픽셀이 입력될 때부터 결과 영상



(그림 5) 카메라에서 획득한 원본 영상
(Fig. 5) Raw image acquired from camera

(그림 6) 동적 문턱치화를 적용한 결과 영상
(Fig. 6) Dynamic thresholding applied image.



(그림 7) 잡음 감소가 포함된 동적 문턱치화 회로
(Fig. 7) Block diagram of dynamic thresholding circuit with cascaded noise reduction circuit

의 첫 픽셀이 출력될 때 가지의 지연 시간은 동적 문턱치화 회로의 지연시간과 잡음 감소 필터의 지연시간을 합한 것과 같다. (그림 5)의 문턱치화 적용 영상에 잡음 감소 필터를 적용한 결과를 (그림 8)에 보인다.

3.3 국부 히스토그램 평활화

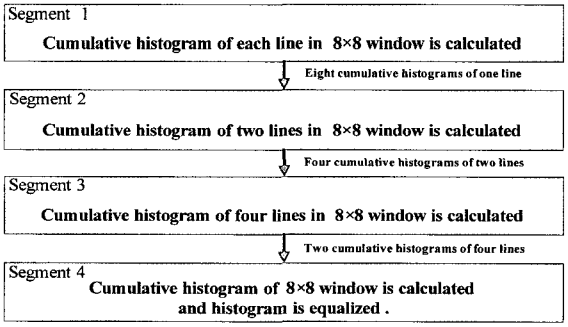
히스토그램 평활화는 영상의 전체적인 명암대비를 향상시키기 위해서 사용된다. 영상의 전체 영역을 기반으로 히스토그램을 계산하여 이를 기반으로 평활화를 수행하는 전역 히스토그램 평활화와 달리, 국부 히스토그램 평활화는 영상을 여러 부분으로 분할하여 각 영역의 히스토그램을 계산하고 이를 이용하여 각 영역에 평활화를 수행한다. 이렇게 함으로써 영상 일부에 밝기 변화가 있더라도 적절한 평활화를 적용하는 것이 가능해진다. 중첩 블록(Block-overlapped) 히스토그램 평활화는 다양한 국부 히스토그램 평활화한 방법이다[10]. 중첩 블록 히스토그램 평활화는 한 픽셀에 대해 해당 픽셀과 특정 수의 주변 픽셀에 대한 히스토그램을 계산하고 이를 이용하여 해당 픽셀에 대해 평활화를 수행한다. 이때 포함되는 주위 픽셀의 수에 따라 결과 영상이 영향을 받는데, 너무 많은 픽셀을 포함 시키면 영상 내의 국부 영역에 대한 밝기 변화에 대해 평활화를 바르게 수행하기 어

(그림 8) 잡음이 감소된 동적 문턱치화 영상
(Fig. 8) Noise reduction applied dynamic thresholding image

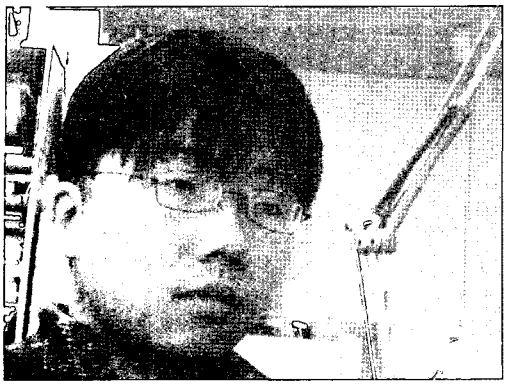
렵고, 너무 적은 픽셀만을 포함시키면 영상 전체적인 히스토그램의 특성을 충분히 반영하지 못해 적절한 평활화를 수행하기 힘들다.

3.4 국부 히스토그램 평활화 구현

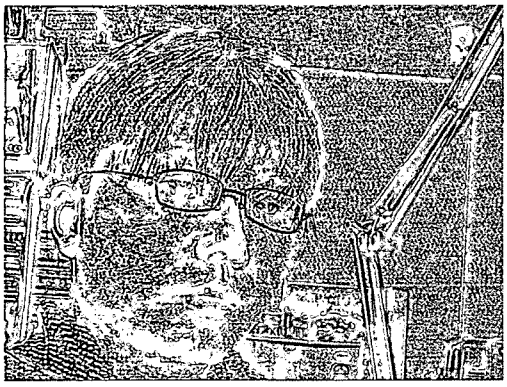
본 논문에서는 제안한 구조를 사용하여 *8×8* 윈도우를 기반으로 하는 중첩 블록 히스토그램 평활화 회로를 구현하였다. 국부 히스토그램 평활화 연산은 누적 히스토그램 생성과 평활화를 포함하여 비교적 복잡한 단계를 거친다. 따라서 픽셀의 입력 주기 내에 연산을 마치기 힘들기 때문에 연산 파이프라인의 세그먼트를 나누어줄 필요가 있다. 수행한 실험에서는 픽셀 주기와 히스토그램 생성 방법의 특징을 고려하여 (그림 9)에 나타낸 것처럼 4개의 세그먼트로 나누어 구성 하였다. 각 세그먼트가 수행하는 연산은 다음과 같다. 세그먼트 1은 *8×8* 윈도우를 구성하는 각 라인의 누적 히스토그램(cumulative histogram)을 생성하여 총 8개의 누적 히스토그램을 만들어 낸다. 세그먼트 2와 3은 각각 이전 단계의 세그먼트가 생성한 누적 히스토그램을 두 개씩 모아서 새로운 누적 히스토그램을 생성한다. 마지막으로 세그먼트 4가 이를 사용하여 *8×8* 윈도우 전체에 대한 누적 히스토그램을 생성하고 전제적인 평활화를 수행한다. (그림 10)에 나타난 원본 실험 영상에 국부 히스토그램 평활화를 수행하면 (그림 11)와 같은 결과 영상을 얻을 수 있다.



(그림 9) 중첩 블록 히스토그램 평활화 파이프라인
(Fig. 9) Block diagram of block-overlapped histogram equalization processing pipeline



(그림 10) 카메라에서 획득한 어두운 원본 영상
(Fig. 10) Raw dark image acquired from camera



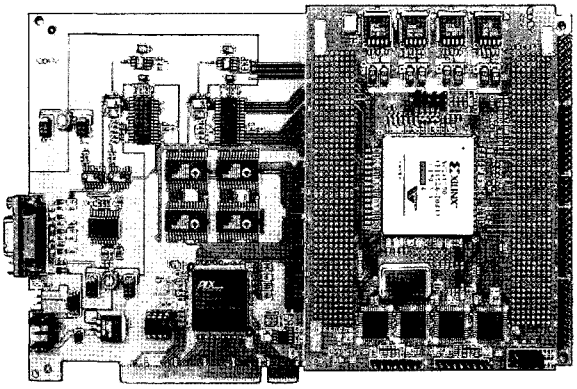
(그림 11) 국부 히스토그램 평활화 결과 영상
(Fig. 11) Local histogram equalization applied image

4. 실험 결과

(그림 12)에 본 논문에서 제안한 윈도우 기반의 영상처리를 구현한FPGA 기반의 영상 처리 시스템을 보인다. 제안한 전체 구조는 Xilinx사의 XC2V6000-FF1152 FPGA상에 구현되었다. 이 시스템은 표준 RS-170 카메라에서 640x480의 그레이 영상을 받아서 처리한 후 원래 영상과 처리한 결과 영상을 PCI 버스를 통해서 PC로 전송한다.

구현 결과를 비교하기 위해 바이패스 회로, 동적 문턱화 회로, 잡음 감소가 포함된 동적 문턱화 회로 그리고 중첩 블록 히스토그램 평활화 회로의 합성 결과를 <표 1>에 보인다. <표 2>에서는 각 구현에 대해서 제안한 구조를 사용한 하드웨어 회로의 수행 시간과 범용 컴퓨터에서 프로그램을 실행하여 얻은 수행 시간을 비교한 결과를 보인다. 실험을 위해 사용한 소프트웨어 프로그램은 Intel Pentium 4 CPU(2.4 GHz)에 1GB DDR SDRAM PC2100(266 MHz) 메모리를 사용한 PC에서 Microsoft Windows XP professional 운영체제 하에서 Microsoft Visual C++ 6.0으로 컴파일하고 실행 결과를 얻었다. 성능 비교를 위해 프로그램은 500번 반복 수행한 후 평균 시간을 계산하여 비교하였다.

<표 2>에서 나타낸 것과 같이 제안한 하드웨어 구조와 비교했을 때 범용 컴퓨터 상에서 윈도우 기반의 영상처리를 수행하는 데에는 많은 처리 시간이 소요된다는 것을 알 수 있다. 표준 RS-170 카메라는 1초당 30 프레임의 영상을 출력하기 때문에 범용 컴퓨터에서 이를 실시간으로 처리하기는 쉽지 않다. 반면에 제안한 병렬 하드웨어 구조를 사용한 회로는 초당 30 프레임 이상의 정보를 실시간으로 처리 할 수 있음을 알 수 있다.



(그림 12) FPGA기반의 영상 처리 시스템
(Fig. 12) FPGA-based image processing system

〈표 1〉 합성결과
〈Table 1〉 Synthesis result

A. 바이패스 회로
A. Bypass circuit

Complete architecture	990 slices
FPGA percentage	1%
BRAMs percentage	4%
Maximum frequency	128.974 MHz

B. 동적 문턱화 회로
B. Dynamic thresholding circuit

Complete architecture	1210 slices
FPGA percentage	2%
BRAMs percentage	4%
Maximum frequency	45.701 MHz

C. 잡음 감소가 포함된 동적 문턱화 회로
C. Dynamic threshold circuit with noise reduction

Complete architecture	1361 slices
FPGA percentage	2 %
BRAMs percentage	6 %
Maximum frequency	45.113 MHz

D. 국부 히스토그램 평활화 회로
D. Local histogram equalization circuit

Complete architecture	3883 slices
FPGA percentage	8%
BRAMs percentage	11%
Maximum frequency	16.576 MHz

〈표 2〉 성능 비교
〈Table 2〉 Performance comparison.

A. 동적 문턱화 회로
A. Dynamic thresholding circuit

	HW circuit**	SW program***
Window size	5×5	5×5
Processing time*	0.008 sec	0.121 sec
Maximum fps	112.511 fps	8.264 fps

B. 잡음 감소가 포함된 동적 문턱화 회로
B. Dynamic threshold circuit with noise reduction

	HW circuit**	SW program***
Window size	5×5 and 3×3	5×5 and 3×3
Processing time*	0.008 sec	0.132 sec
Maximum fps	111.307 fps	7.575 fps

C. 국부 히스토그램 평활화 회로
C. Local histogram equalization circuit

	HW circuit**	SW program***
Window size	8×8	8×8
Processing time*	0.024 sec	1.315 sec
Maximum fps	40.898 fps	0.760 fps

* 1 frame에 대한 연산 시간
** VHDL Implementation
*** Visual C++ Implementation

5. 결 론

본 논문에서는 윈도우 기반의 영상처리를 실시간으로 수행할 수 있는 병렬 하드웨어 구조를 제안하였다. 또한 이 구조를 사용하여 동적 문턱치화 회로와 국부 히스토그램 평활화 회로를 구현하여 다른 윈도우 기반의 영상처리 알고리즘이 제안한 구조를 통해 쉽게 구현 될 수 있음을 보였다. 일반적으로 범용 컴퓨터를 사용하여 구현된 시스템에서 제안한 구조의 회로를 사용하면 상위 레벨에 추가적인 부하를 주지 않고 전처리 영상을 실시간으로 획득하는 것이 가능하다. 본 논문에서 제안한 구조는 향후 영상처리 SoC(System on a Chip) 설계에 응용될 것이다.

참 고 문 헌

[1] R.C. Gonzalez and R.E. Woods, *Digital Image Processing,* Prentice-hall, New Jersey, 2nd Edition, 2002.

[2] N. Ranganathan, *VLSI & Parallel Computing for Pattern Recognition & Artificial Intelligence,* Series in Machine Perception and Artificial Intelligence, Volume 18, World Scientific Publishing, 1995.

[3] C. Torres-Huitzil and M. Arias-Estrada, "Configurable Hardware Architecture for Real-Time Window-Based Image Processing," *Lecture Notes in Computer Science, Springer-Verlag,* Vol.2778, pp.1008-1011, 2003.

[4] J. Woodfill, G. Gordon and R. Buck, "Tyzx DeepSea High Speed Stereo Vision System," *Conference on Computer Vision and Pattern Recognition,* pp.41-45, June, 2004.

[5] R. Jain, R.Kasturi, B.G. Schunck, *Machine Vision,* McGraw-Hill, Singapore, International Edition, 1995.

[6] A.K. Jain, *Fundamentals of Digital Image Processing, Prentice-hall,* Englewood Cliffs, 1989.

[7] D. Woitha, D. Janich, "Adaptive Threshold," *Lecture note in computer science, Springer-Verlag,* No.2959, 2004.

[8] Nir Milstein, "Image Segmentation by Adaptive Thresholding," Spring 1998, http://www.cs.technion.ac.il /Labs/Isl/Project/Projects_done/Visionclasses/Vision_1998/Adaptive_Thresholding/

[9] C.W. Kurak Jr., "Adaptive Histogram Equalization A Parallel Implementation," *Computer-Based Medical Systems: Fourth Annual IEEE Symposium, Track 2: Medical Imaging Systems/Session* 11, 1991

[10] J.-Y. Kim, L.-S. Kim, S.-H, Hwang, "An Adaptive Contrast Enhancement Using Partially Overlapped Sub-Block Histogram Equalization," *IEEE Transactions on Circuit and Systems for Video Technology,* Vol.11, No.4, April, 2001.

진 승 훈

e-mail : coredev@ece.skku.ac.kr
2005년 성균관대학교 정보통신공학부 (학사)
2005년~현재 성균관대학교 전자전기 공학과 석사과정
관심분야 : SoC, 컴퓨터 비전, 내장형 시스템

조 정 욱

e-mail : ichead@ece.skku.ac.kr
2001년 성균관대학교 전기전자 및 컴퓨터 공학부(학사)
2003년 성균관대학교 전기전자 및 컴퓨터 공학과(공학석사)
2006년 성균관대학교 전자전기공학과 (공학박사)
2006년~현재 성균관대학교 정보통신공학부 연구교원
관심분야 : 모션제어기, 비전 시스템, 내장형 시스템

권 기 호

e-mail : khkwon@yurim.skku.ac.kr
1975년 성균관대학교 전자공학과(학사)
1978년 서울대학교 전자공학과(공학석사)
1989년 성균관대학교 전자전기공학과 (공학박사)
1989년~현재 성균관대학교 정보통신공학부 교수
관심분야 : 퍼지이론, 유전자 알고리즘, 카오스이론

전 재 욱

e-mail : jwjeon@yurim.skku.ac.kr
1984년 서울대학교 전자공학과(학사)
1986년 서울대학교 전자공학과(공학석사)
1990년 Purdue University (Ph.D)
1990년~1994년 삼성전자 생산기술센터 선임연구원
1994년~현재 성균관대학교 정보통신공학부 교수
관심분야 : 로봇공학, 내장형 시스템, 공장 자동화