SystemC를 이용한 Edge Detector의 구현

오영진, 송기용
충북대학교 컴퓨터공학과

Implementation of an Edge Detector with SystemC

Young-Jin Oh, Gi-Yong Song
Dept. of Computer Engineering, Chungbuk National University

요 약

본 논문에서는 시스템 수준 설계 언어인 SystemC를 이용하여 디지털 이미지 프로세싱의 한 부분인 Edge Detector 구현에 대하여 기술한다. Edge Detector는 마스크의 가운데 픽셀을 강조하는 Sobel 알고리즘을 사용하여 모델링 하였으며, 설계된 디자인의 동작은 간단한 BMP 파일을 적용하여 검증하였다. 또한 Edge Detector를 구성하는 하위 모듈들 간의 연결을 각각 sc_buffer 채널과 sc_fifo 채널을 이용하여 설계하였을 때의 실행시간을 비교하였다.

I. 서론

최근 시스템의 설계가 대형화되고 복잡해지면서 시스템 수준에서의 설계가 중요하게 되었다. 이러한 현실에 부응하여 high-level abstraction에 기반하여 시스템을 모델링하고 명시하는 SystemC와 같은 시스템 수준 설계 언어가 대두되고 있다.

SystemC는 표준 C/C++ 언어를 기반으로 시스템 레벨의 설계를 지원하는 설계 환경을 라이브러리 형태로 제공하는 언어이다 [1-3]. 추가된 라이브러리는 C/C++ 언어가 제공하지 못하는 하드웨어 및 시스템 모델링을 위한 몇 가지 클래스가 포함하여 time, hardware data type, concurrency 등의 개념을 지원하고 있다. 상위 수준에서 설계된 SystemC 모델은 시뮬레이션과 구체화 합성의 설계 과정을 하나의 언어로 진행 시킬 수 있어 설계상의 시간이 단축되고 점진적인 개선을 통하여 합성 가능한 수준의 시스템을 구현할 수 있다 [1].

SystemC는 빠른 시뮬레이션이 가능하도록 사건-기반 시뮬레이션 커널을 제공한다. 또한 모듈과 프로세스 사이의 통신을 위해서 채널, 인터페이스, 포트 그리고 이벤트와 같은 개념을 포함한다.

본 논문에서는 SystmeC를 이용하여 Edge-Detector를 설계한 후 간단한 BMP 파일을 가지고 그 동작을 검증하였으며, 또한 Edge Detector를 구성하는 하위 모듈들 간의 연결을 각각 sc_buffer 채널과 sc_fifo 채널을 이용하여 설계하여 그 실행시간을 비교하였다. 설계된 디자인은 Sobel 알고리즘을 사용하였으며, Booth multiplier를 사용하여 2D 컨벌루션의 곱셈 연산을 수행한다.

II. SystemC

SystemC는 하드웨어 모델링용 라이브러리를 포함시킨 C++ 클래스 라이브러리와 사건-기반 시뮬레이션 커널로 구성된 시스템 수준 설계 언어로 high-level abstraction을 이용한 하드웨어 소자, 소프트웨어 모듈, 그리고 이 둘이 결합된 시스템을 co-design 할 수 있는 설계 환경을 제공한다. SystemC는 병렬성을 프로세스로 모델링하고, 시스템의 구조는 계층구조를 지원하는 모듈(module)과 포트(port), 그리고 채널(channel)등으로 모델링하며, 시간 개념의 클럭을 지원한다 [1-4].

SystemC를 이용한 설계 흐름도를 그림 1에 보인다 [1]. SystemC는 시스템 수준에서 모델을 설계하기 시작해서 점진적인 설계 구체화를 통해 RTL 수준까지 단일 언어 환경에서 설계할 수 있도록 해준다. 시스템 설계의 최종 단계는 결국 단위 모듈의 통합이라고 볼 때, 시스템 모델링과 사양 설정의 초기 단계에서부터 구현까지 점진적인 추상화 수준 변환이 가능한 단일 언어의 설계 환경을 갖는다는 것은 큰 장점이라 할 수 있다.

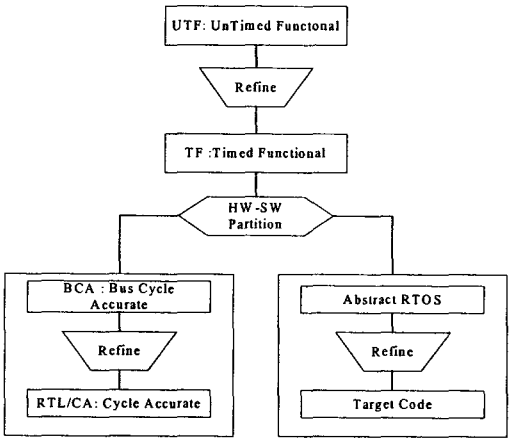

그림1. SystemC 설계 흐름도

SystemC는 모듈이라고 불리는 컨테이너 클래스를 사용하기 위하여 SC_MODULE이라는 키워드를 사용한다. 모듈은 내부에 다른 모듈이나 프로세스를 가질 수 있기 때문에 계층적인 구조를 가지며, 모듈의 동작은 프로세스를 추상화한 SC_METHOD, SC_THREAD, SC_CTHREAD를 사용하여 기술한다. 프로세스는 인터페이스, 채널, 포트를 통하여 다른 프로세서와 상호 통신하며 이벤트로 동기화 된다. 포트는 모듈 내부와 외부의 정보를 전달하고 채널로 포트와 포트 사이를 연결한다 [2,3]. 인터페이스는 채널에 대한 일련의 접근 방법이 선언되어 있어 포트는 이것을 통해 채널과 연결된다. SystemC에서 제공하는 기본 채널, 인터페이스, 포트를 표 1에 보인다.

표 1. 기본 채널

기본채널	인터페이스	method	포트	method
sc_signal<T>	sc_signal_inout_if<T>	read() write()	sc_in<T>	read()
sc_buffer<T>			sc_out<T>	write()
			sc_inout<T>	Read() write()
sc_fifo<T>	sc_fifo_in_if<T>	read() nb_read()	sc_fifo_in<T>	read() nb_read()
	sc_fifo_out_if<T>	rrite() nb_write()	sc_fifo_out<T>	write() nb_write()
sc_clock	sc_signal_if<boo>			

III. 2D 컨벌루션과 Edge Detector

본 논문에서는 Sobel 알고리즘이 사용된 Edge Detector 시스템을 SystemC를 이용하여 설계한다. Sobel 알고리즘은 이미지 프로세싱의 에지 검출을 위한 전형적인 마스킹 방법이다 [6,7]. 설계된 디자인은 2D 컨벌루션의 곱셈을 Booth multiplier를 사용하여 수행한다.

1. 2D 컨벌루션

컨벌루션은 이미지 프로세싱에서 이미지를 강조하기 위해 요구되는 기본적인 연산의 하나로 smoothing, sharpness, denoising, edge detection 등의 선형 필터링 연산에서 사용된다 [6]. 일반적으로 이미지 프로세싱은 2차원의 공간/배열의 형태로 옮겨져 2차원 공간의 수의 배열로 나타난다. 컨벌루션은 입력 픽셀의 이웃에 있는 픽셀들의 가중치 합이으로 가중치는 작은 2차원 배열로 주어진다. 이 배열은 흔히 컨벌루션 마스크 또는 컨벌루션 커널이라고 한다 [6,7]. 가중치 마스크는 대개 5×5 또는 3×3의 크기를 갖는 2차원 배열로 만들어 진다. 여기서는 Sobel 알고리즘을 위해 3×3 마스크를 사용한다.

식 (1)은 이미지 프로세싱을 위한 2D 컨벌루션을 나타낸다 [5].

$$OI[x,y] = FC[x,y] \otimes IP[x,y] \qquad (1)$$
$$= \sum_{I=0}^{n-1}\sum_{J=0}^{n-1} FC[I,J] \times IP[x-I+(\frac{n-1}{2}), y-J+(\frac{n-1}{2})]$$

식 (1)에서 IP는 입력 이미지 부분, FC는 필터 계수 부분, OI는 출력 이미지 부분을 각각 의미한다. IP와 OI는 $i \times j$ 픽셀의 크기를 가지며 FC는 $n \times n$ 픽셀의 크기를 갖는다. 대부부의 경우 $n<i$, $n<j$ 이다 [5].

2개의 연속적인 출력 픽셀이 만들어지는 예를 그림 2에 보인다. 모든 IP 픽셀은 FC 픽셀과 서로 겹쳐지고 대응하는 FC 픽셀과 곱셈 연산을 한다. 곱셈의 합이 컨벌루션의 결과로 하나의 출력 픽셀 OI[x,y]의 값이 된다. FC는 OI[3,4]을 계산하기 위해 IP[3,4]를 가운데에 위치시켜 계산하고, OI[3,5]를 위해 IP[3,5]로 이동한다.

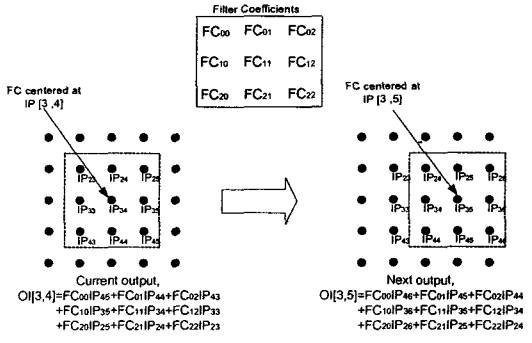

그림 2. 컨벌루션 연산

2. Edge Detector

Edge는 그레이 수준의 속성을 갖는 2개 영역 사이의 테두리이다 [6].

Sobel 필터는 이미지의 방향에 상관없이 마스크의 가운데 픽셀을 강조하는 특성을 가지며 수직 또는 수평 방향보다 대각선 방향에 더 영향을 받는다 [6,7]. Sobel 필터의 이러한 속성은 두 방향의 에지 강조 연산의 합으로

구현된다. Sobel 알고리즘은 다음과 같은 과정을 따른다 [6].

단계 1 : 공간 컨벌루션 마스크 1을 정의한다.

단계 2 : 화소 집단 처리

단계 3 : 공간 컨벌루션 마스크 2를 정의한다.

단계 4 : 화소 집단 처리

단계 5 : 복수 이미지에 대한 화소 단위 처리 - 덧셈

Sobel 에지 강조 연산 결과가 0보다 작거나 255보다 큰 경우, 결과를 각각 0 또는 255로 만든다. Sobel 에지 마스크를 표 2에 보인다.

표 2. Sobel 에지 마스크

-1	0	1
-2	0	2
-1	0	1

Verticality mask

-1	-2	-1
0	0	0
1	2	1

Horizontality mask

Ⅳ. SystemC를 이용한 시스템 모델링

Edge Detector는 4단계로 수행된다.

단계 1 : 입력 이미지의 헤더 파일과 RGB 데이터를 읽는다.

단계 2 : 각 픽셀의 RGB 데이터를 그레이 수준의 이미지 데이터로 변경한다.

단계 3 : Sobel 알고리즘을 이용하여 그레이 수준의 데이터에 대한 에지 검출을 수행한다.

단계 4 : 출력 이미지 파일을 작성한다.

Edge Detector의 내부 구조를 그림 3에 보인다.

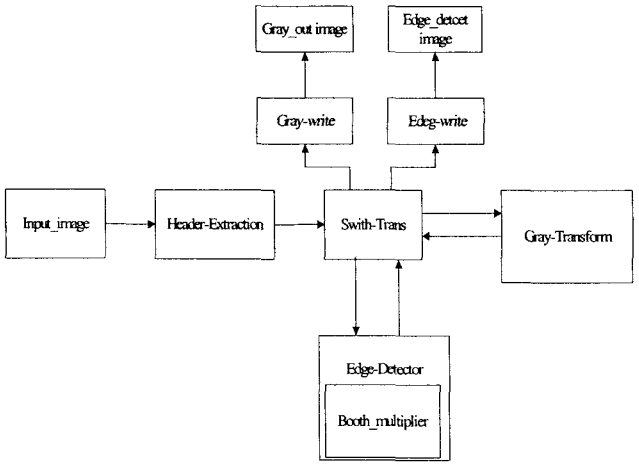

그림 3. Edge Detector의 내부 구조

1. BMP 파일

이미지는 GIF(Graphics Interchange Format), JPEG(Joint Photographic coding Experts Group) 또는 BMP(Bit MaP)와 같은 파일 형식으로 저장된다. 본 논문에서 설계하는 Edge Detector의 입력 이미지는 표준 비트맵 그래픽 형식의 BMP 파일 형식 이미지로 제한된다. BMP 파일의 이미지는 필요에 따라 2, 16, 또는 256가지의 색으로 저장된다. 식 (2)는 BMP 파일의 크기를 계산하는 식이다 [8].

$$SIZE=(width\ in\ pixels)\times(height\ in\ pixels)+54 \quad (2)$$

54라는 수는 BMP 파일 헤더의 크기이다. BMP 파일의 헤더는 이미지 파일의 모든 정보를 포함하며 그림 4에서 헤더의 구조를 보인다. 하나의 사각형은 1바이트를 의미하며 각각의 정보들은 big-endian 방식으로 저장된다. *bfSize*는 4 바이트로 비트맵 파일의 전체 크기를 나타낸다. *bfOffBits*는 4바이트 크기로 저장되며 이미지의 RGB 데이터의 파일 오프셋을 나타낸다. 4바이트 크기인 *biWidth*, *biHeight*, 그리고 *biSizeImage*는 각각 비트맵 이미지의 가로 픽셀 수, 세로 필셀 수, 그리고 파일 헤더 데이터를 제외한 이미지의 전체 크기를 나타낸다.

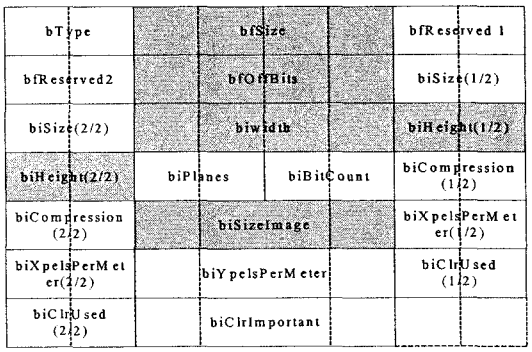

그림 4. 비트맵 파일의 헤더 구조

2. 모델링

Edge Detector를 구성하는 7개 모듈의 기능은 다음과 같다.

Header-Extraction 모듈 : 입력 BMP 파일을 열고 헤더 데이터를 읽어 들여 이미지의 전체 크기와 가로의 크기, 세로의 크기와 같은 중요 데이터를 발췌한 후, 헤더 데이터와 RGB 이미지 데이터를 Switch-Trans 모듈로 전달한다.

Switch-Trans 모듈 : 다음과 같은 순서로 모듈 간 정보의 전달만을 수행한다.

1. 그레이 수준의 이미지 데이터를 얻기 위해 Header-Extraction 모듈로부터 입력받은 데이터를 Gray-Transform 모듈로 전달한다.
2. 에지 검출을 위해 Gray-Transform 모듈로부터 입력

받은 그레이 수준 정보를 Edge-Detector 모듈로 전달한다.

3. 출력 BMP 파일을 얻기 위해 Edge-Detector 모듈로부터 받은 결과 데이터를 Gray-Write 모듈과 Edge-Write 모듈로 전달한다.

Gray-Transform 모듈 : 각 픽셀의 RGB 이미지 데이터에 대하여 평균값을 취하는 방법으로 그레이 수준의이미지 데이터로 변환한다.

Edge-Detector 모듈 : 하위 모듈인 Booth-multiplier 모듈을 이용하여 Sobel 알고리즘의 2D 컨벌루션을 수행한다.

Booth-multiplier 모듈 : 2D 컨벌루션의 곱셈연산을 Booth 알고리즘을 이용하여 수행한다.

Gray-Write 모듈 : Gray-Transform 모듈에 의해 만들어진 그레이 수준 이미지를 파일로 저장한다.

Edge-Write 모듈 : Edge-Detector 모듈에 의해 만들어진 에지 강조 이미지를 파일로 저장한다.

그림 3에서 각 모듈의 연결을 각각 sc_buffer 채널과 sc_fifo 채널을 이용하여 설계하였을 때의 실행시간을 표 3에 보인다. 모듈 간의 연결을 sc_buffer 채널을 이용하여 Edge Detector를 설계하였을 경우 RGB 데이터를 그레이 수준의 데이터로 변환 과정에서 약 2초, 에지 강조 연산을 수행하는 과정에서 약 20초, 결과 데이터를 파일로 출력하는데 약 1초의 시간이 걸린다. sc_fifo 채널을 이용하여 모듈 간의 연결을 한 경우는 fifo 크기를 6으로 하여 블럭킹 방식으로 설계하였다. 수행 시간은 3초가 걸리며 sc_buffer 채널을 사용하는 것보다 속도 측면에서 약 7.6배 빨라짐을 확인할 수 있다.

표3. 채널별 수행시간

	수행시간 (초)
sc_buffer	23
sc_fifo	3

그림 5의 원본 이미지를 설계한 Edge Detector에 입력으로 주었을 때의 결과를 그림 6과 그림 7에 각각 보인다.

그림 5. 원본 이미지

그림 6. Gray 이미지

그림 7. edge detection 이미지

V. 결론

본 논문에서는 SystemC를 이용하여 행위 수준으로 Edge Detector를 구현하였다. 설계된 디자인은 그림 3에 보이는 것처럼 계층적인 구조를 갖는 7개의 모듈로 구성되며, IV절에서 언급한 4단계의 과정으로 동작을 수행한다. 전체 시스템은 하나의 최상위 레벨 모듈로 구현되고 간단한 BMP 파일을 통해 설계한 Edge Detector의 동작을 검증하였다.

향후 연구과제는 Edge Detector를 BMP 형식뿐만 아니라 JPEG 또는 다른 이미지 파일 형식에 대해서도 동작하도록 설계하는 것이다.

참 고 문 헌

[1] 기안도, *SystmeC 시스템모델링 언어*, 대영사, 2004

[2] J.Bhasker, A *SystemC Primer*, Star Galaxy Publishing, 2002

[3] David C. Black, Jack Donovan, *SystemC:From The Ground Up*, Eklectic Ally, Inc., 2004

[4] Thorsten Grotker,Stan Liao, Grant Martin, Stuart Swan. *System Design with SystemC*, Kluwer Academic Publishers, Boston, 2003.

[5] Albert Tung-Hoe Wong, *"A New Scalable Systolic Array Processor Architecture for Discrete Convolution"*, The graduate school, University of Kentucky, 2003

[6] Greory A. Baxes, *Digital image processing : principles and applications*, John WILEY & Sons., Inc., 1994

[7] Rafael C. Gonzalez, Richard E. Woods, *Digital image processing*, Addison-Wesley, 1993

[8] S.Y.Lee, *Visual C++ Programming Bible ver.6.x*, youngjin.com, Inc., Korea, 2000

[9] http://www.systemc.org